

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

IN RE APPLICATION OF: Kunihiro AKIYOSHI, et al.

GAU: 2622

SERIAL NO: 10/627,731

EXAMINER:

FILED: July 28, 2003

FOR: IMAGE FORMING APPARATUS, INFORMATION PROCESSING APPARATUS AND VERSION CHECK METHOD

REQUEST FOR PRIORITY

COMMISSIONER FOR PATENTS
ALEXANDRIA, VIRGINIA 22313

SIR:

- ☐ Full benefit of the filing date of U.S. Application Serial Number , filed , is claimed pursuant to the provisions of 35 U.S.C. §120.
- ☐ Full benefit of the filing date(s) of U.S. Provisional Application(s) is claimed pursuant to the provisions of 35 U.S.C. §119(e):
Application No. Date Filed
- ☒ Applicants claim any right to priority from any earlier filed applications to which they may be entitled pursuant to the provisions of 35 U.S.C. §119, as noted below.

In the matter of the above-identified application for patent, notice is hereby given that the applicants claim as priority:

<u>COUNTRY</u>	<u>APPLICATION NUMBER</u>	<u>MONTH/DAY/YEAR</u>
Japan	2002-224136	July 31, 2002
Japan	2003-195193	July 10, 2003
Japan	2003-195194	July 10, 2003

Certified copies of the corresponding Convention Application(s)

- ☒ are submitted herewith
- ☐ will be submitted prior to payment of the Final Fee
- ☐ were filed in prior application Serial No. filed
- ☐ were submitted to the International Bureau in PCT Application Number
Receipt of the certified copies by the International Bureau in a timely manner under PCT Rule 17.1(a) has been acknowledged as evidenced by the attached PCT/IB/304.
- ☐ (A) Application Serial No.(s) were filed in prior application Serial No. filed ; and
- ☐ (B) Application Serial No.(s)
☐ are submitted herewith
☐ will be submitted prior to payment of the Final Fee

Respectfully Submitted,

OBLON, SPIVAK, McCLELLAND,
MAIER & NEUSTADT, P.C.


Marvin J. Spivak

Registration No. 24,913

Joseph A. Scafetta, Jr.
Registration No. 26,803

Customer Number

22850

Tel. (703) 413-3000
Fax. (703) 413-2220
(OSMMN 05/03)

10/627,731

日 本 国 特 許 庁

JAPAN PATENT OFFICE

65

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出 願 年 月 日

Date of Application:

2002年 7月31日

出 願 番 号

Application Number:

特願2002-224136

[ST.10/C]:

[JP2002-224136]

出 願 人

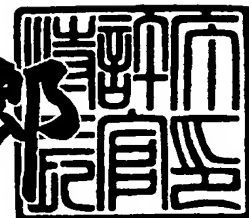
Applicant(s):

株式会社リコー

2003年 5月 6日

特 許 庁 長 官
Commissioner,
Japan Patent Office

太田 信一郎



出証番号 出証特2003-3032441

【書類名】 特許願

【整理番号】 0203940

【提出日】 平成14年 7月31日

【あて先】 特許庁長官殿

【国際特許分類】 G03G 21/00 370

【発明の名称】 画像形成装置およびバージョンチェック方法

【請求項の数】 27

【発明者】

 【住所又は居所】 東京都大田区中馬込 1 丁目 3 番 6 号 株式会社リコー内

 【氏名】 秋吉 邦洋

【発明者】

 【住所又は居所】 東京都大田区中馬込 1 丁目 3 番 6 号 株式会社リコー内

 【氏名】 安藤 光男

【特許出願人】

 【識別番号】 000006747

 【氏名又は名称】 株式会社リコー

【代理人】

 【識別番号】 100089118

 【弁理士】

 【氏名又は名称】 酒井 宏明

【手数料の表示】

 【予納台帳番号】 036711

 【納付金額】 21,000円

【提出物件の目録】

 【物件名】 明細書 1

 【物件名】 図面 1

 【物件名】 要約書 1

 【包括委任状番号】 9808514

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 画像形成装置およびバージョンチェック方法

【特許請求の範囲】

【請求項 1】 画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションと、前記アプリケーションと画像形成処理で使用されるハードウェア資源との間に介在し、前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスとを備えた画像形成装置であって、

前記アプリケーションのインストール時から起動後の実行時までの間に、前記アプリケーションが使用する関数単位毎のバージョン情報を取得すると共に、前記コントロールサービスをサーバとしたクライアントプロセスとして動作し、かつ前記アプリケーションをクライアントとしたサーバプロセスとして動作する仮想アプリケーションサービスのインターフェイスの関数単位毎のバージョン情報を取得し、対応する関数のバージョンが同じか否かをチェックする関数単位バージョンチェック手段

を備えたことを特徴とする画像形成装置。

【請求項 2】 前記仮想アプリケーションサービスは、前記アプリケーション毎に設けられ、前記各アプリケーションを個別に仮起動または通常起動させることを特徴とする請求項 1 に記載の画像形成装置。

【請求項 3】 前記コントロールサービスは、前記ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うことを特徴とする請求項 1 または 2 に記載の画像形成装置。

【請求項 4】 前記関数単位バージョンチェック手段による関数単位毎のバージョンチェックを行う前に、前記各アプリケーションの全体バージョン情報を取得すると共に、前記仮想アプリケーションサービスの全体バージョン情報を取得し、全体バージョンが同じか否かをチェックする全体バージョンチェック手段をさらに備え、

前記全体バージョンチェック手段による比較結果に基づいて全体バージョンが

異なっている場合のみ、前記関数単位バージョンチェック手段による関数単位毎のバージョンチェックを行うことを特徴とする請求項1～3のいずれか一つに記載の画像形成装置。

【請求項5】 前記アプリケーションが使用する関数単位毎のバージョン情報と前記仮想アプリケーションサービスのインターフェースの関数単位毎のバージョン情報、および、前記アプリケーションの全体バージョン情報と前記仮想アプリケーションサービスの全体バージョン情報の少なくとも一方のバージョン情報を取得するバージョン情報取得手段をさらに備えたことを特徴とする請求項1～4のいずれか一つに記載の画像形成装置。

【請求項6】 前記バージョン情報取得手段は、予め必要なバージョン情報を持たせた実行ファイルを用いて、前記アプリケーションの仮起動、あるいは、通常起動の際にプロセス間通信を行うことにより、所望のバージョン情報を取得することを特徴とする請求項5に記載の画像形成装置。

【請求項7】 前記バージョン情報取得手段は、前記アプリケーションの実行ファイルを生成する際に、前記アプリケーションが使用する関数単位毎のバージョン情報を記載した使用関数テーブルを生成すると共に、前記仮想アプリケーションサービスの実行ファイルを生成する際に、前記仮想アプリケーションサービスのインターフェースの関数単位毎のバージョン情報を記載した全関数テーブルを生成し、前記使用関数テーブルと前記全関数テーブルのいずれか一方のバージョン情報を他方へ通知することにより、比較可能なバージョン情報を取得することを特徴とする請求項5に記載の画像形成装置。

【請求項8】 前記関数単位バージョンチェック手段は、前記アプリケーションのインストール時、前記アプリケーションの起動時、および、前記アプリケーションの起動後の実行時のいずれかの時点でバージョンチェックを行うことを特徴とする請求項1～7のいずれか一つに記載の画像形成装置。

【請求項9】 前記関数単位バージョンチェック手段が前記アプリケーションのインストール時にバージョンチェックを行う場合は、前記仮想アプリケーションサービスが前記アプリケーションを仮起動させて、当該アプリケーションの使用関数単位毎のバージョン情報が取得できるようにしたことを特徴とする

請求項 8 に記載の画像形成装置。

【請求項 1 0】 前記関数単位バージョンチェック手段が前記アプリケーションの起動時にバージョンチェックを行う場合は、前記仮想アプリケーションサービスが前記アプリケーションを通常起動させて、当該アプリケーションの使用
する関数単位毎のバージョン情報が取得できるようにしたことを特徴とする請求
項 8 に記載の画像形成装置。

【請求項 1 1】 前記関数単位バージョンチェック手段が前記アプリケーションの起動後の実行時にバージョンチェックを行う場合は、前記仮想アプリケーションサービスが前記アプリケーションを通常起動させた際に取得した、当該ア
プリケーションの使用する関数単位毎のバージョン情報に基づいて、その後実行
される関数単位毎のバージョンをチェックすることを特徴とする請求項 8 に記載
の画像形成装置。

【請求項 1 2】 前記関数単位バージョンチェック手段、前記全体バージョ
ンチェック手段、および、前記バージョン情報取得手段の少なくとも 1 つは、前
記仮想アプリケーションサービスに含まれていることを特徴とする請求項 1 ～ 1
1 のいずれか一つに記載の画像形成装置。

【請求項 1 3】 前記関数単位バージョンチェック手段、前記全体バージョ
ンチェック手段、および、前記バージョン情報取得手段の少なくとも 1 つは、前
記仮想アプリケーションサービスのプロセス内部でスレッドとして生成されるこ
とを特徴とする請求項 1 2 に記載の画像形成装置。

【請求項 1 4】 前記関数単位バージョンチェック手段、前記全体バージョ
ンチェック手段、および、前記バージョン情報取得手段の少なくとも 1 つは、前
記アプリケーションに含まれることを特徴とする請求項 1 ～ 1 1 のいずれか一つ
に記載の画像形成装置。

【請求項 1 5】 前記関数単位バージョンチェック手段、前記全体バージョ
ンチェック手段、および、前記バージョン情報取得手段の少なくとも 1 つは、前
記アプリケーションのプロセス内部でスレッドとして生成されることを特徴とす
る請求項 1 4 に記載の画像形成装置。

【請求項 1 6】 画像形成処理にかかるユーザサービスにそれぞれ固有の処

理を行うアプリケーションと、前記アプリケーションと画像形成処理で使用されるハードウェア資源との間に介在し、前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスとを備えた画像形成装置上で、前記アプリケーションとの整合性をチェックするバージョンチェック方法であって、

前記アプリケーションのインストール時から起動後の実行時までの間に、前記アプリケーションが使用する関数単位毎のバージョン情報を取得すると共に、前記コントロールサービスをサーバとしたクライアントプロセスとして動作し、かつ前記アプリケーションをクライアントとしたサーバプロセスとして動作する仮想アプリケーションサービスのインターフェイスの関数単位毎のバージョン情報を取得し、対応する関数のバージョンが同じか否かをチェックする関数単位バージョンチェックステップ

を含むことを特徴とするバージョンチェック方法。

【請求項 1 7】 前記コントロールサービスは、前記ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うことを特徴とする請求項 1 6 に記載のバージョンチェック方法。

【請求項 1 8】 前記関数単位バージョンチェックステップによる関数単位毎のバージョンチェックを行う前に、前記各アプリケーションの全体バージョン情報を取得すると共に、前記仮想アプリケーションサービスの全体バージョン情報を取得し、全体バージョンが同じか否かをチェックする全体バージョンチェックステップをさらに含み、

前記全体バージョンチェックステップによる比較結果に基づいて全体バージョンが異なっている場合のみ、前記関数単位バージョンチェックステップによる関数単位毎のバージョンチェックを行うことを特徴とする請求項 1 6 または 1 7 に記載のバージョンチェック方法。

【請求項 1 9】 前記アプリケーションが使用する関数単位毎のバージョン情報と前記仮想アプリケーションサービスのインターフェイスの関数単位毎のバージョン情報、および、前記アプリケーションの全体バージョン情報と前記仮想

アプリケーションサービスの全体バージョン情報の少なくとも一方のバージョン情報を取得するバージョン情報取得ステップをさらに含むことを特徴とする請求項 1 6 ～ 1 8 のいずれか一つに記載のバージョンチェック方法。

【請求項 2 0】 前記バージョン情報取得ステップは、予め必要なバージョン情報を持たせた実行ファイルを用いて、前記アプリケーションの仮起動、あるいは、通常起動の際にプロセス間通信を行うことにより、所望のバージョン情報を取得することを特徴とする請求項 1 9 に記載のバージョンチェック方法。

【請求項 2 1】 前記バージョン情報取得ステップは、前記アプリケーションの実行ファイルを生成する際に、前記アプリケーションが使用する関数単位毎のバージョン情報を記載した使用関数テーブルを生成すると共に、前記仮想アプリケーションサービスの実行ファイルを生成する際に、前記仮想アプリケーションサービスのインターフェイスの関数単位毎のバージョン情報を記載した全関数テーブルを生成し、前記使用関数テーブルと前記全関数テーブルのいずれか一方のバージョン情報を他方へ通知することにより、比較可能なバージョン情報を取得することを特徴とする請求項 1 9 に記載のバージョンチェック方法。

【請求項 2 2】 前記関数単位バージョンチェックステップは、前記アプリケーションのインストール時、前記アプリケーションの起動時、および、前記アプリケーションの起動後の実行時のいずれかの時点でバージョンチェックを行うことを特徴とする請求項 1 6 ～ 2 1 のいずれか一つに記載のバージョンチェック方法。

【請求項 2 3】 前記関数単位バージョンチェックステップで前記アプリケーションのインストール時にバージョンチェックを行う場合は、前記仮想アプリケーションサービスが前記アプリケーションを仮起動させて、当該アプリケーションの使用する関数単位毎のバージョン情報が取得できるようにしたことを特徴とする請求項 2 2 に記載のバージョンチェック方法。

【請求項 2 4】 前記関数単位バージョンチェックステップで前記アプリケーションの起動時にバージョンチェックを行う場合は、前記仮想アプリケーションサービスが前記アプリケーションを通常起動させて、当該アプリケーションの使用する関数単位毎のバージョン情報が取得できるようにしたことを特徴とする

請求項 2 2 に記載のバージョンチェック方法。

【請求項 2 5】 前記関数単位バージョンチェックステップで前記アプリケーションの起動後の実行時にバージョンチェックを行う場合は、前記仮想アプリケーションサービスが前記アプリケーションを通常起動させた際に取得した、当該アプリケーションの使用する関数単位毎のバージョン情報に基づいて、その後実行される関数単位毎のバージョンをチェックすることを特徴とする請求項 2 2 に記載のバージョンチェック方法。

【請求項 2 6】 前記関数単位バージョンチェックステップ、前記全体バージョンチェックステップ、および、前記バージョン情報取得ステップの少なくとも 1 つの処理が前記仮想アプリケーションサービスで行われることを特徴とする請求項 1 6 ～ 2 5 のいずれか一つに記載のバージョンチェック方法。

【請求項 2 7】 前記関数単位バージョンチェックステップ、前記全体バージョンチェックステップ、および、前記バージョン情報取得ステップの少なくとも 1 つの処理が前記アプリケーションで行われることを特徴とする請求項 1 6 ～ 2 5 のいずれか一つに記載のバージョンチェック方法。

【発明の詳細な説明】

【0 0 0 1】

【発明の属する技術分野】

この発明は、コピー、プリンタ、スキャナおよびファクシミリなどの画像形成処理にかかるユーザサービスを提供するアプリケーションのバージョン不整合による不具合発生を未然に防止することができる画像形成装置およびバージョンチェック方法に関するものである。

【0 0 0 2】

【従来の技術】

近年では、プリンタ、コピー、ファクシミリ、スキャナなどの各装置の機能を 1 つの筐体内に収納した画像形成装置（以下、「複合機」という。）が知られている。この複合機は、1 つの筐体内に表示部、印刷部および撮像部などを設けるとともに、プリンタ、コピーおよびファクシミリ装置にそれぞれ対応した 3 種類のソフトウェアを設け、これらのソフトウェアを切り替えることによって、当該

装置をプリンタ、コピー、スキャナまたはファクシミリ装置として動作させるものである。

【0003】

このような従来の複合機では、プリンタ、コピー、ファクシミリ、スキャナなどの各機能単位でアプリケーションプログラムが起動され、ハードウェア資源にアクセスする機能を持ち合わせている。その際、アプリケーションプログラムとオペレーティングシステム（OS）のバージョンが同じことが前提となるが、例えば、OSがバージョンアップしてアプリケーションとの間でバージョン差が生じた場合、今まで使えていた機能が使えなくなったり、アプリケーションそのものが起動しなくなったりすることがある。

【0004】

このため、従来の複合機では、OSがバージョンアップされた場合、それに伴ってアプリケーションをコンパイルし直すなどして、常に整合性のとれたバージョン関係にあることが要請されている。

【0005】

【発明が解決しようとする課題】

ところで、このような従来の複合機では、プリンタ、コピー、スキャナおよびファクシミリ装置に対応するソフトウェアをそれぞれ別個に設けているため、各ソフトウェアの開発に多大の時間を要する。このため、出願人は、表示部、印刷部および撮像部などの画像形成処理で使用するハードウェア資源を有し、プリンタ、コピーまたはファクシミリなどの各ユーザサービスにそれぞれ固有の処理を行うアプリケーションを複数搭載し、これらのアプリケーションとハードウェア資源との間に介在して、ユーザサービスを提供する際に、アプリケーションの少なくとも2つが共通的に必要とするハードウェア資源の管理、実行制御並びに画像形成処理を行う各種コントロールサービスからなるプラットフォームを備えた画像形成装置（複合機）を発明した。

【0006】

このような新規な複合機では、アプリケーションとコントロールサービスが別個に設けられているため、複合機の出荷後にユーザもしくは第三者であるサード

ベンダが新規なアプリケーションを開発して複合機に搭載することが可能であり、これによって多種多様な機能を提供することが可能となっている。

【0007】

このような新規な複合機では、アプリケーションの少なくとも2つが共通的に必要とするサービスを提供するコントロールサービスがアプリケーションと別個に設けられているので、新規アプリケーションを開発する場合、各種コントロールサービスとのプロセス間通信を実現する処理をソースコードで記述する必要がある。しかしながら、新規アプリケーションを開発する場合は、各コントロールサービスが提供する関数やメッセージなどを正確に把握した上で、予め規定された手順で記述しなければならない、各コントロールサービスやアプリケーションからの処理要求を受信可能とするアプリケーションプログラムインタフェース（API）がデバッグや機能追加によってバージョンアップが繰り返されると、ベンダーはどのバージョンに合わせてアプリケーション開発をすればよいか非常に分かり難くなり、アプリケーションの開発自体が阻害されるおそれが出てきた。このことは、固定的な複数の機能を寄せ集めた従来の複合機では問題にならなかった新規な課題である。

【0008】

また、各コントロールサービスとアプリケーションのインタフェース（API）のすべてを、新規アプリケーションを開発するサードベンダなどの第三者に開示することは、プログラムの秘匿性の面から言って好ましくない。特に、複合機のシステムに大きく影響を与えるような箇所については、第三者に対して隠蔽することにより、直接コントロールサービスに対してアクセスされるのを回避することが可能となり、複合機のセキュリティ面および障害発生未然防止の面からも好ましい。

【0009】

そこで、出願人は、上記複合機のコントロールサービスとアプリケーションとの間に一定のサポート範囲でバージョン差を吸収して整合性をとると共に、アプリケーションとコントロールサービスとの間の複雑なインタフェースを隠蔽して重要なインタフェースの秘匿性を確保しつつ、特定のインタフェースを開示して

新規アプリケーションの開発効率を向上させることのできる仮想アプリケーションサービス（V A S）を追加したさらに新規な複合機を発明した。この仮想アプリケーションサービス（V A S）は、コントロールサービスをサーバとしたクライアントプロセスとして動作し、かつ、アプリケーションをクライアントとしたサーバプロセスとして動作するものである。

【 0 0 1 0 】

しかしながら、このような新規な複合機における仮想アプリケーションサービス（V A S）のバージョンは、常にコントロールサービス側と整合性がとれるようにバージョンアップが繰り返されると、今度はアプリケーションと仮想アプリケーションサービス（V A S）との間でバージョンの不整合が生じ易くなり、実行時（ランタイム時）におけるエラーの発生が問題となる可能性が出てきた。

【 0 0 1 1 】

そこで、アプリケーションと仮想アプリケーションサービス（V A S）のバージョンの整合性がとれているか否かを事前にチェックして、ランタイム時におけるエラー発生を防止するため、全体のバージョン同士を比較して、一致していれば整合性有りとし、一致していない場合は整合性無しとすることも可能である。しかし、これでは僅かなバージョン差が生じても整合性無しとされるため、バージョン差の吸収範囲があまりにも狭く、コントロールサービス側でバージョンアップがあると使用できなくなるアプリケーションが多数発生するおそれがあった。

【 0 0 1 2 】

この発明は上記に鑑みてなされたもので、コントロールサービス側のバージョンアップに伴って仮想アプリケーションサービスがバージョンアップされても、アプリケーションとの間に生じたバージョン差をできるだけ吸収して、利用可能なアプリケーションを増やすと共に、アプリケーションのランタイム時におけるエラーの発生を確実に防止することができる画像形成装置およびバージョンチェック方法を得ることを目的とする。

【 0 0 1 3 】

【課題を解決するための手段】

上記目的を達成するため、請求項 1 にかかる発明は、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションと、前記アプリケーションと画像形成処理で使用するハードウェア資源との間に介在し、前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスとを備えた画像形成装置であって、前記アプリケーションのインストール時から起動後の実行時までの間に、前記アプリケーションが使用する関数単位毎のバージョン情報を取得すると共に、前記コントロールサービスをサーバとしたクライアントプロセスとして動作し、かつ前記アプリケーションをクライアントとしたサーバプロセスとして動作する仮想アプリケーションサービスのインターフェイスの関数単位毎のバージョン情報を取得し、対応する関数のバージョンが同じか否かをチェックする関数単位バージョンチェック手段を備えたことを特徴とする。

【 0 0 1 4 】

この請求項 1 にかかる発明によれば、関数単位バージョンチェック手段によって、アプリケーションのインストール時から起動後の実行時までの間に、アプリケーションが使用する関数単位毎のバージョン情報を取得すると共に、コントロールサービスをサーバとしたクライアントプロセスとして動作し、かつアプリケーションをクライアントとしたサーバプロセスとして動作する仮想アプリケーションサービスのインターフェイスの関数単位毎のバージョン情報を取得して、対応する関数のバージョンが同じか否かをチェックするようにしたため、アプリケーションが使用していない関数に対応した仮想アプリケーションサービスのインターフェイスの関数に変更、追加、あるいは削除されてバージョンアップしたとしても、両者間の整合性に影響を与えないことから、全体バージョン同士を単純比較する場合と比べて、バージョン差の吸収範囲が広くなり、利用可能なアプリケーションを増やすことができる。また、アプリケーションが実際に使用する関数単位毎のバージョン同士を個別に比較するため、確実にランタイム時におけるエラー発生を防止することができる。

【 0 0 1 5 】

また、請求項 2 にかかる発明は、請求項 1 に記載の画像形成装置において、前

記仮想アプリケーションサービスは、前記アプリケーション毎に設けられ、前記各アプリケーションを個別に仮起動または通常起動させることを特徴とする。

【 0 0 1 6 】

この請求項 2 にかかかる発明によれば、仮想アプリケーションサービスは、アプリケーション毎に設けられ、各アプリケーションを個別に仮起動または通常起動させるようにしたため、各アプリケーションが使用する関数単位毎のバージョン情報を取得し、これに対応した仮想アプリケーションサービスのインターフェースの関数単位毎のバージョン情報を取得して、対応する関数同士のバージョンチェックを容易に行うことができる。

【 0 0 1 7 】

また、請求項 3 にかかかる発明は、請求項 1 または 2 に記載の画像形成装置において、前記コントロールサービスは、前記ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うことを特徴とする。

【 0 0 1 8 】

この請求項 3 にかかかる発明によれば、コントロールサービスは、ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とするハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うようにしたため、複数のアプリケーションを作成する際に、共通化できる部分をコントロールサービス（プラットフォーム）として共通化することにより、それ以外の部分を作成すればよく、アプリケーションの開発が容易となり、かつ、短期間で開発することができる。

【 0 0 1 9 】

また、請求項 4 にかかかる発明は、請求項 1 ～ 3 のいずれか一つに記載の画像形成装置において、前記関数単位バージョンチェック手段による関数単位毎のバージョンチェックを行う前に、前記各アプリケーションの全体バージョン情報を取得すると共に、前記仮想アプリケーションサービスの全体バージョン情報を取得し、全体バージョンが同じか否かをチェックする全体バージョンチェック手段をさらに備え、前記全体バージョンチェック手段による比較結果に基づいて全体バ

ージョンが異なっている場合のみ、前記関数単位バージョンチェック手段による関数単位毎のバージョンチェックを行うことを特徴とする。

【0020】

この請求項4にかかる発明によれば、アプリケーションの全体バージョン情報と仮想アプリケーションサービスの全体バージョン情報とを取得して、アプリケーションと仮想アプリケーションサービスの全体バージョン同士を比較する全体バージョンチェック手段をさらに備えていて、全体バージョンが異なっている場合にのみ関数単位バージョンチェック手段による関数単位毎のバージョンチェックを行うようにしたため、アプリケーションが使用する関数単位毎の個別のバージョンチェックを常に行う必要がなくなり、バージョンチェック処理を効率良く、かつ迅速に行うことができる。

【0021】

また、請求項5にかかる発明は、請求項1～4のいずれか一つに記載の画像形成装置において、前記アプリケーションが使用する関数単位毎のバージョン情報と前記仮想アプリケーションサービスのインターフェ이스の関数単位毎のバージョン情報、および、前記アプリケーションの全体バージョン情報と前記仮想アプリケーションサービスの全体バージョン情報の少なくとも一方のバージョン情報を取得するバージョン情報取得手段をさらに備えたことを特徴とする。

【0022】

この請求項5にかかる発明によれば、バージョン情報取得手段により、アプリケーションが使用する関数単位毎のバージョン情報と仮想アプリケーションサービスのインターフェ이스の関数単位毎のバージョン情報を取得したり、あるいは、アプリケーションの全体バージョン情報と仮想アプリケーションサービスの全体バージョン情報を取得することができるため、取得したバージョン情報同士を比較するだけで、容易にバージョンチェックを行うことができる。

【0023】

また、請求項6にかかる発明は、請求項5に記載の画像形成装置において、前記バージョン情報取得手段は、予め必要なバージョン情報を持たせた実行ファイルを用いて、前記アプリケーションの仮起動、あるいは、通常起動の際にプロセ

ス間通信を行うことにより、所望のバージョン情報を取得することを特徴とする。

【 0 0 2 4 】

この請求項 6 にかかる発明によれば、バージョン情報取得手段は、予め必要なバージョン情報を持たせた実行ファイルを使って、アプリケーションを仮起動、あるいは、通常起動させた際のプロセス間通信により、所望のバージョン情報を取得するようにしたため、バージョン情報を容易に一元管理することができる。

【 0 0 2 5 】

また、請求項 7 にかかる発明は、請求項 5 に記載の画像形成装置において、前記バージョン情報取得手段は、前記アプリケーションの実行ファイルを生成する際に、前記アプリケーションが使用する関数単位毎のバージョン情報を記載した使用関数テーブルを生成すると共に、前記仮想アプリケーションサービスの実行ファイルを生成する際に、前記仮想アプリケーションサービスのインターフェイスの関数単位毎のバージョン情報を記載した全関数テーブルを生成し、前記使用関数テーブルと前記全関数テーブルのいずれか一方のバージョン情報を他方へ通知することにより、比較可能なバージョン情報を取得することを特徴とする。

【 0 0 2 6 】

この請求項 7 にかかる発明によれば、バージョン情報取得手段は、アプリケーションの実行ファイルの生成時にアプリケーションが使用する関数毎のバージョン情報が記載された使用関数テーブルを生成し、仮想アプリケーションサービスの実行ファイルの生成時に仮想アプリケーションサービスのインターフェイスの関数毎のバージョン情報が記載された全関数テーブルを生成して、使用関数テーブルと全関数テーブルのいずれか一方のバージョン情報を他方へ通知することにより、比較可能なバージョン情報が取得できるため、アプリケーションを仮起動や通常起動させることなくバージョンチェックを行うことができる。

【 0 0 2 7 】

また、請求項 8 にかかる発明は、請求項 1 ～ 7 のいずれか一つに記載の画像形成装置において、前記関数単位バージョンチェック手段は、前記アプリケーションのインストール時、前記アプリケーションの起動時、および、前記アプリケー

ションの起動後の実行時のいずれかの時点でバージョンチェックを行うことを特徴とする。

【 0 0 2 8 】

この請求項 8 にかかる発明によれば、関数単位バージョンチェック手段は、アプリケーションのインストール時、アプリケーションの起動時、および、アプリケーションの起動後の実行時のいずれかの時点でバージョンチェックを行うようにしたため、アプリケーションのインストール状況等に応じて関数単位毎のバージョンチェックを適切な時期に行うことが可能となり、アプリケーションのバージョン不整合によるランタイム時のエラー発生を未然に防ぐことができる。

【 0 0 2 9 】

また、請求項 9 にかかる発明は、請求項 8 に記載の画像形成装置において、前記関数単位バージョンチェック手段が前記アプリケーションのインストール時にバージョンチェックを行う場合は、前記仮想アプリケーションサービスが前記アプリケーションを仮起動させて、当該アプリケーションの使用する関数単位毎のバージョン情報が取得できるようにしたことを特徴とする。

【 0 0 3 0 】

この請求項 9 にかかる発明によれば、関数単位バージョンチェック手段がアプリケーションのインストール時にバージョンチェックする場合、仮想アプリケーションサービスがアプリケーションを仮起動させて、そのアプリケーションが使用する関数単位毎のバージョン情報を得ることができるため、未インストールの新規アプリケーションをインストールする際に、仮想アプリケーションサービスとの間のバージョンの整合性をチェックすることができる。このように、インストールされていないアプリケーションであっても仮起動させることにより、アプリケーションと仮想アプリケーションサービスとの間でプロセス間通信が可能となるため、各種情報を双方向でやり取りすることが可能となる。

【 0 0 3 1 】

また、請求項 1 0 にかかる発明は、請求項 8 に記載の画像形成装置において、前記関数単位バージョンチェック手段が前記アプリケーションの起動時にバージョンチェックを行う場合は、前記仮想アプリケーションサービスが前記アプリケ

ーションを通常起動させて、当該アプリケーションの使用する関数単位毎のバージョン情報が取得できるようにしたことを特徴とする。

【 0 0 3 2 】

この請求項 1 0 にかかる発明によれば、関数単位バージョンチェック手段がアプリケーションの起動時にバージョンチェックする場合、仮想アプリケーションサービスがアプリケーションを通常起動させて、そのアプリケーションが使用する関数単位毎のバージョン情報を得ることができるため、インストール済みのアプリケーション、あるいは、未インストールのアプリケーションをとりあえずインストールしておいて、通常起動の際にバージョンチェックを受けることができる。この場合は、アプリケーションを通常起動させるため、アプリケーションと仮想アプリケーションサービスとの間でプロセス間通信を行うことが可能であり、各種情報を双方向でやり取りすることができる。

【 0 0 3 3 】

また、請求項 1 1 にかかる発明は、請求項 8 に記載の画像形成装置において、前記関数単位バージョンチェック手段が前記アプリケーションの起動後の実行時にバージョンチェックを行う場合は、前記仮想アプリケーションサービスが前記アプリケーションを通常起動させた際に取得した、当該アプリケーションの使用する関数単位毎のバージョン情報に基づいて、その後実行される関数単位毎のバージョンをチェックすることを特徴とする。

【 0 0 3 4 】

この請求項 1 1 にかかる発明によれば、関数単位バージョンチェック手段がアプリケーションの起動後の実行時にバージョンチェックする場合、仮想アプリケーションサービスがアプリケーションを通常起動させた際に取得した当該アプリケーションの使用する関数単位毎のバージョン情報に基づいて、その後実行される関数単位毎のバージョンをチェックするようにしたため、アプリケーションの起動後の実行時においてバージョンチェックを受けることができる。この場合は、既にアプリケーションを通常起動させているため、アプリケーションと仮想アプリケーションサービスとの間でのプロセス間通信が可能であり、各種情報を双方向でやり取りすることができる。

【 0 0 3 5 】

また、請求項 1 2 にかかる発明は、請求項 1 ～ 1 1 のいずれか一つに記載の画像形成装置において、前記関数単位バージョンチェック手段、前記全体バージョンチェック手段、および、前記バージョン情報取得手段の少なくとも 1 つは、前記仮想アプリケーションサービスに含まれていることを特徴とする。

【 0 0 3 6 】

この請求項 1 2 にかかる発明によれば、関数単位バージョンチェック手段、全体バージョンチェック手段、および、バージョン情報取得手段の少なくとも 1 つが仮想アプリケーションサービスに含まれているため、バージョンチェック処理を主に仮想アプリケーションサービス側で行うことができる。

【 0 0 3 7 】

また、請求項 1 3 にかかる発明は、請求項 1 2 に記載の画像形成装置において、前記関数単位バージョンチェック手段、前記全体バージョンチェック手段、および、前記バージョン情報取得手段の少なくとも 1 つは、前記仮想アプリケーションサービスのプロセス内部でスレッドとして生成されることを特徴とする。

【 0 0 3 8 】

この請求項 1 3 にかかる発明によれば、関数単位バージョンチェック手段、全体バージョンチェック手段、および、バージョン情報取得手段の少なくとも 1 つが仮想アプリケーションサービスのプロセス内部でスレッドとして生成されるため、複数のアプリケーションが並列に起動されたり、バージョン情報取得処理と関数単位バージョンチェック処理と全体バージョンチェック処理とを、コンテキスト切り替えなしに CPU 占有切り替えによる並列実行を行うことが可能となり、これらの並列処理を円滑に行うことができる。

【 0 0 3 9 】

また、請求項 1 4 にかかる発明は、請求項 1 ～ 1 1 のいずれか一つに記載の画像形成装置において、前記関数単位バージョンチェック手段、前記全体バージョンチェック手段、および、前記バージョン情報取得手段の少なくとも 1 つは、前記アプリケーションに含まれることを特徴とする。

【 0 0 4 0 】

この請求項 1 4 にかかる発明によれば、関数単位バージョンチェック手段、全体バージョンチェック手段、および、バージョン情報取得手段の少なくとも 1 つがアプリケーションに含まれているため、バージョンチェック処理を主にアプリケーション側で行うことができる。

【 0 0 4 1 】

また、請求項 1 5 にかかる発明は、請求項 1 4 に記載の画像形成装置において、前記関数単位バージョンチェック手段、前記全体バージョンチェック手段、および、前記バージョン情報取得手段の少なくとも 1 つは、前記アプリケーションのプロセス内部でスレッドとして生成されることを特徴とする。

【 0 0 4 2 】

この請求項 1 5 にかかる発明によれば、関数単位バージョンチェック手段、全体バージョンチェック手段、および、バージョン情報取得手段の少なくとも 1 つがアプリケーションのプロセス内部でスレッドとして生成されるため、バージョン情報取得処理と関数単位バージョンチェック処理と全体バージョンチェック処理とを、コンテキスト切り替えなしに CPU 占有切り替えによる並列実行を行うことが可能となり、これらの並列処理を円滑に行うことができる。

【 0 0 4 3 】

また、請求項 1 6 にかかる発明は、画像形成処理にかかるユーザサービスにそれぞれ固有の処理を行うアプリケーションと、前記アプリケーションと画像形成処理で使用されるハードウェア資源との間に介在し、前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うコントロールサービスとを備えた画像形成装置上で、前記アプリケーションとの整合性をチェックするバージョンチェック方法であって、前記アプリケーションのインストール時から起動後の実行時までの間に、前記アプリケーションが使用する関数単位毎のバージョン情報を取得すると共に、前記コントロールサービスをサーバとしたクライアントプロセスとして動作し、かつ前記アプリケーションをクライアントとしたサーバプロセスとして動作する仮想アプリケーションサービスのインターフェイスの関数単位毎のバージョン情報を取得し、対応する関数のバージョンが同じか否かをチェックする関数単位バージョンチェックステップを含むことを特徴とする。

【 0 0 4 4 】

この請求項 1 6 にかかる発明によれば、関数単位バージョンチェックステップによって、アプリケーションのインストール時から起動後の実行時までの間に、アプリケーションが使用する関数単位毎のバージョン情報を取得すると共に、コントロールサービスをサーバとしたクライアントプロセスとして動作し、かつアプリケーションをクライアントとしたサーバプロセスとして動作する仮想アプリケーションサービスのインターフェイスの関数単位毎のバージョン情報を取得して、対応する関数のバージョンが同じか否かをチェックするようにしたため、アプリケーションが使用していない関数に対応した仮想アプリケーションサービスのインターフェイスの関数に変更、追加、あるいは削除されてバージョンアップしたとしても、両者間の整合性に影響を与えないことから、全体バージョン同士を単純比較する場合と比べて、バージョン差の吸収範囲が広くなり、利用可能なアプリケーションを増やすことができる。また、アプリケーションが実際に使用する関数単位毎のバージョン同士を個別に比較するため、確実にランタイム時におけるエラー発生を防止することができる。

【 0 0 4 5 】

また、請求項 1 7 にかかる発明は、請求項 1 6 に記載のバージョンチェック方法において、前記コントロールサービスは、前記ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とする前記ハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うことを特徴とする。

【 0 0 4 6 】

この請求項 1 7 にかかる発明によれば、コントロールサービスは、ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とするハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うようにしたため、複数のアプリケーションを作成する際に、共通化できる部分をコントロールサービス（プラットフォーム）として共通化することにより、それ以外の部分を作成すればよく、アプリケーションの開発が容易となり、かつ、短期間で開発することができる。

【 0 0 4 7 】

また、請求項 1 8 にかかる発明は、請求項 1 6 または 1 7 に記載のバージョンチェック方法において、前記関数単位バージョンチェックステップによる関数単位毎のバージョンチェックを行う前に、前記各アプリケーションの全体バージョン情報を取得すると共に、前記仮想アプリケーションサービスの全体バージョン情報を取得し、全体バージョンが同じか否かをチェックする全体バージョンチェックステップをさらに含み、前記全体バージョンチェックステップによる比較結果に基づいて全体バージョンが異なっている場合のみ、前記関数単位バージョンチェックステップによる関数単位毎のバージョンチェックを行うことを特徴とする。

【 0 0 4 8 】

この請求項 1 8 にかかる発明によれば、アプリケーションの全体バージョン情報と仮想アプリケーションサービスの全体バージョン情報とを取得して、アプリケーションと仮想アプリケーションサービスの全体バージョン同士を比較する全体バージョンチェックステップをさらに含んでいて、全体バージョンが異なっている場合にのみ関数単位バージョンチェックステップによる関数単位毎のバージョンチェックを行うようにしたため、アプリケーションが使用する関数単位毎の個別のバージョンチェックを常に行う必要がなくなり、バージョンチェック処理を効率良く、かつ迅速に行うことができる。

【 0 0 4 9 】

また、請求項 1 9 にかかる発明は、請求項 1 6 ～ 1 8 のいずれか一つに記載のバージョンチェック方法において、前記アプリケーションが使用する関数単位毎のバージョン情報と前記仮想アプリケーションサービスのインターフェイスの関数単位毎のバージョン情報、および、前記アプリケーションの全体バージョン情報と前記仮想アプリケーションサービスの全体バージョン情報の少なくとも一方のバージョン情報を取得するバージョン情報取得ステップをさらに含むことを特徴とする。

【 0 0 5 0 】

この請求項 1 9 にかかる発明によれば、バージョン情報取得ステップによって、アプリケーションが使用する関数単位毎のバージョン情報と仮想アプリケーシ

ョンサービスのインターフェイスの関数単位毎のバージョン情報を取得したり、あるいは、アプリケーションの全体バージョン情報と仮想アプリケーションサービスの全体バージョン情報を取得することができるため、取得したバージョン情報同士を比較するだけで、容易にバージョンチェックを行うことができる。

【 0 0 5 1 】

また、請求項 2 0 にかかる発明は、請求項 1 9 に記載のバージョンチェック方法において、前記バージョン情報取得ステップは、予め必要なバージョン情報を持たせた実行ファイルを用いて、前記アプリケーションの仮起動、あるいは、通常起動の際にプロセス間通信を行うことにより、所望のバージョン情報を取得することを特徴とする。

【 0 0 5 2 】

この請求項 2 0 にかかる発明によれば、バージョン情報取得ステップは、予め必要なバージョン情報を持たせた実行ファイルを使って、アプリケーションを仮起動、あるいは、通常起動させた際のプロセス間通信により、所望のバージョン情報を取得するようにしたため、バージョン情報を容易に一元管理することができる。

【 0 0 5 3 】

また、請求項 2 1 にかかる発明は、請求項 1 9 に記載のバージョンチェック方法において、前記バージョン情報取得ステップは、前記アプリケーションの実行ファイルを生成する際に、前記アプリケーションが使用する関数単位毎のバージョン情報を記載した使用関数テーブルを生成すると共に、前記仮想アプリケーションサービスの実行ファイルを生成する際に、前記仮想アプリケーションサービスのインターフェイスの関数単位毎のバージョン情報を記載した全関数テーブルを生成し、前記使用関数テーブルと前記全関数テーブルのいずれか一方のバージョン情報を他方へ通知することにより、比較可能なバージョン情報を取得することを特徴とする。

【 0 0 5 4 】

この請求項 2 1 にかかる発明によれば、バージョン情報取得ステップは、アプリケーションの実行ファイルの生成時にアプリケーションが使用する関数毎のバ

ージョン情報が記載された使用関数テーブルを生成し、仮想アプリケーションサービスの実行ファイルの生成時に仮想アプリケーションサービスのインターフェイスの関数毎のバージョン情報が記載された全関数テーブルを生成して、使用関数テーブルと全関数テーブルのいずれか一方のバージョン情報を他方へ通知することにより、比較可能なバージョン情報が取得できるため、アプリケーションを仮起動や通常起動させることなくバージョンチェックを行うことができる。

【 0 0 5 5 】

また、請求項 2 2 にかかる発明は、請求項 1 6 ～ 2 1 のいずれか一つに記載のバージョンチェック方法において、前記関数単位バージョンチェックステップは、前記アプリケーションのインストール時、前記アプリケーションの起動時、および、前記アプリケーションの起動後の実行時のいずれかの時点でバージョンチェックを行うことを特徴とする。

【 0 0 5 6 】

この請求項 2 2 にかかる発明によれば、関数単位バージョンチェックステップは、アプリケーションのインストール時、アプリケーションの起動時、および、アプリケーションの起動後の実行時のいずれかの時点でバージョンチェックを行うようにしたため、アプリケーションのインストール状況等に応じて関数単位毎のバージョンチェックを適切な時期に行うことが可能となり、アプリケーションのバージョン不整合によるランタイム時のエラー発生を未然に防ぐことができる。

【 0 0 5 7 】

また、請求項 2 3 にかかる発明は、請求項 2 2 に記載のバージョンチェック方法において、前記関数単位バージョンチェックステップで前記アプリケーションのインストール時にバージョンチェックを行う場合は、前記仮想アプリケーションサービスが前記アプリケーションを仮起動させて、当該アプリケーションの使用関数単位毎のバージョン情報が取得できるようにしたことを特徴とする。

【 0 0 5 8 】

この請求項 2 3 にかかる発明によれば、関数単位バージョンチェックステップでアプリケーションのインストール時にバージョンチェックする場合、仮想アプ

リケーションサービスがアプリケーションを仮起動させて、そのアプリケーションが使用する関数単位毎のバージョン情報を得ることができるため、未インストールの新規アプリケーションをインストールする際に、仮想アプリケーションサービスとの間のバージョンの整合性をチェックすることができる。このように、インストールされていないアプリケーションであっても仮起動させることにより、アプリケーションと仮想アプリケーションサービスとの間でプロセス間通信が可能となるため、各種情報を双方向でやり取りすることが可能となる。

【0059】

また、請求項24にかかる発明は、請求項22に記載のバージョンチェック方法において、前記関数単位バージョンチェックステップで前記アプリケーションの起動時にバージョンチェックを行う場合は、前記仮想アプリケーションサービスが前記アプリケーションを通常起動させて、当該アプリケーションの使用する関数単位毎のバージョン情報が取得できるようにしたことを特徴とする。

【0060】

この請求項24にかかる発明によれば、関数単位バージョンチェックステップでアプリケーションの起動時にバージョンチェックする場合、仮想アプリケーションサービスがアプリケーションを通常起動させて、そのアプリケーションが使用する関数単位毎のバージョン情報を得ることができるため、インストール済みのアプリケーション、あるいは、未インストールのアプリケーションをとりあえずインストールしておいて、通常起動の際にバージョンチェックを受けることができる。この場合は、アプリケーションを通常起動させるため、アプリケーションと仮想アプリケーションサービスとの間でプロセス間通信を行うことが可能であり、各種情報を双方向でやり取りすることができる。

【0061】

また、請求項25にかかる発明は、請求項22に記載のバージョンチェック方法において、前記関数単位バージョンチェックステップで前記アプリケーションの起動後の実行時にバージョンチェックを行う場合は、前記仮想アプリケーションサービスが前記アプリケーションを通常起動させた際に取得した、当該アプリケーションの使用する関数単位毎のバージョン情報に基づいて、その後実行され

る関数単位毎のバージョンをチェックすることを特徴とする。

【 0 0 6 2 】

この請求項 2 5 にかかる発明によれば、関数単位バージョンチェックステップでアプリケーションの起動後の実行時にバージョンチェックする場合、仮想アプリケーションサービスがアプリケーションを通常起動させた際に取得した当該アプリケーションの使用する関数単位毎のバージョン情報に基づいて、その後実行される関数単位毎のバージョンをチェックするようにしたため、アプリケーションの起動後の実行時においてバージョンチェックを受けることができる。この場合は、既にアプリケーションを通常起動させているため、アプリケーションと仮想アプリケーションサービスとの間でのプロセス間通信が可能であり、各種情報を双方向でやり取りすることができる。

【 0 0 6 3 】

また、請求項 2 6 にかかる発明は、請求項 1 6 ～ 2 5 のいずれか一つに記載のバージョンチェック方法において、前記関数単位バージョンチェックステップ、前記全体バージョンチェックステップ、および、前記バージョン情報取得ステップの少なくとも 1 つの処理が前記仮想アプリケーションサービスで行われることを特徴とする。

【 0 0 6 4 】

この請求項 2 6 にかかる発明によれば、関数単位バージョンチェックステップ、全体バージョンチェックステップ、および、バージョン情報取得ステップの少なくとも 1 つの処理が仮想アプリケーションサービスで行われるため、バージョンチェック処理を主に仮想アプリケーションサービス側で行うことができる。

【 0 0 6 5 】

また、請求項 2 7 にかかる発明は、請求項 1 6 ～ 2 5 のいずれか一つに記載のバージョンチェック方法において、前記関数単位バージョンチェックステップ、前記全体バージョンチェックステップ、および、前記バージョン情報取得ステップの少なくとも 1 つの処理が前記アプリケーションで行われることを特徴とする。

【 0 0 6 6 】

この請求項 2 7 にかかる発明によれば、関数単位バージョンチェックステップ、全体バージョンチェックステップ、および、バージョン情報取得ステップの少なくとも 1 つの処理がアプリケーションで行われるため、バージョンチェック処理を主にアプリケーション側で行うことができる。

【 0 0 6 7 】

【発明の実施の形態】

以下に添付図面を参照して、この発明にかかる画像形成装置およびバージョンチェック方法の好適な実施の形態を詳細に説明する。

【 0 0 6 8 】

(実施の形態 1)

図 1 は、この発明の実施の形態 1 である画像形成装置（以下、「複合機」という）の構成を示すブロック図である。図 1 に示すように、複合機 1 0 0 は、白黒レーザプリンタ（B&W LP）1 0 1 と、カラーレーザプリンタ（Color LP）1 0 2 と、スキャナ、ファクシミリ、ハードディスク、メモリ、ネットワークインタフェースなどのハードウェアリソース 1 0 3 を有するとともに、プラットフォーム 1 2 0 とアプリケーション 1 3 0 と仮想アプリケーションサービス（V A S : V i r t u a l A p p l i c a t i o n S e r v i c e）1 4 0 から構成されるソフトウェア群 1 1 0 とを備えている。

【 0 0 6 9 】

仮想アプリケーションサービス（V A S）1 4 0 は、アプリケーション 1 3 0 とプラットフォーム 1 2 0 の間に配置される。V A S 1 4 0 は、アプリケーション（以下、アプリともいう）1 3 0 の各アプリが初めて登録されるときに、同時に登録処理が行われ、アプリから見るとプラットフォーム 1 2 0 のサービス層として認識され、サービス層から見るとアプリとして認識されるように登録される。この V A S 1 4 0 の基本機能としては、アプリ 1 3 0 とプラットフォーム 1 2 0 との間のバージョン差を吸収して整合性を保つと共に、コントロールサービスからのメッセージを取捨選択してプラットフォーム 1 2 0 を意図的に隠蔽するラッピング機能を備えている。

【 0 0 7 0 】

また、この V A S 1 4 0 の本発明における特徴的な機能としては、プラットフォーム 1 2 0 側のバージョンアップに伴って V A S 1 4 0 がバージョンアップされても、アプリの実行時（ランタイム時）にエラーを出さないようにするため、アプリ 1 3 0 と V A S 1 4 0 のインタフェース間のバージョンチェックを行い、動作保証の有無が事前に確認できるようにしたものである。特に、本発明では、アプリと V A S の全体バージョンが違っていても、整合性のとれる限り動作保証を行うことで、V A S のサポート範囲をできるだけ広げるようにした点に特徴がある。

【 0 0 7 1 】

プラットフォーム 1 2 0 は、アプリケーションからの処理要求を解釈してハードウェア資源の獲得要求を発生させるコントロールサービスと、一または複数のハードウェア資源の管理を行い、コントロールサービスからの獲得要求を調停するシステムリソースマネージャ（SRM）1 2 3 と、汎用 OS 1 2 1 とを有する。

【 0 0 7 2 】

コントロールサービスは、複数のサービスモジュールから形成され、SCS（システムコントロールサービス）1 2 2 と、ECS（エンジンコントロールサービス）1 2 4 と、MCS（メモリコントロールサービス）1 2 5 と、OCS（オペレーションパネルコントロールサービス）1 2 6 と、FCS（ファックスコントロールサービス）1 2 7 と、NCS（ネットワークコントロールサービス）1 2 8 とから構成される。なお、このプラットフォーム 1 2 0 は、予め定義された関数により前記アプリケーション 1 3 0 からの処理要求を受信可能とするアプリケーションプログラムインタフェース（API）を有している。

【 0 0 7 3 】

汎用 OS 1 2 1 は、UNIX（登録商標）などの汎用オペレーティングシステムであり、プラットフォーム 1 2 0 並びにアプリケーション 1 3 0 の各ソフトウェアをそれぞれプロセスとして並列実行する。

【 0 0 7 4 】

SRM 1 2 3 のプロセスは、SCS 1 2 2 とともにシステムの制御およびリソースの管理を行うものである。SRM 1 2 3 のプロセスは、スキャナ部やプリン

タ部などのエンジン、メモリ、HDDファイル、ホストI/O（セントロI/F、ネットワークI/F、IEEE1394 I/F、RS232C I/Fなど）のハードウェア資源を利用する上位層からの要求にしたがって調停を行い、実行制御する。

【0075】

具体的には、このSRM123は、要求されたハードウェア資源の利用が可能であるか（他の要求により利用されていないかどうか）を判断し、利用可能であれば要求されたハードウェア資源が利用可能である旨を上位層に伝える。また、SRM123は、上位層からの要求に対してハードウェア資源の利用スケジューリングを行い、要求内容（例えば、プリンタエンジンにより紙搬送と作像動作、メモリ確保、ファイル生成など）を直接実施している。

【0076】

SCS122のプロセスは、アプリ管理、操作部制御、システム画面表示、LED表示、リソース管理、割り込みアプリ制御などを行う。

【0077】

ECS124のプロセスは、白黒レーザプリンタ（B&W LP）101、カラーレーザプリンタ（Color LP）102、スキャナ、ファクシミリなどからなるハードウェアリソース103のエンジンの制御を行う。

【0078】

MCS125のプロセスは、画像メモリの取得および解放、ハードディスク装置（HDD）の利用、画像データの圧縮および伸張などを行う。

【0079】

FCS127のプロセスは、システムコントローラの各アプリ層からPSTN/I SDN網を利用したファクシミリ送受信、BKM（バックアップSRAM）で管理されている各種ファクシミリデータの登録/引用、ファクシミリ読みとり、ファクシミリ受信印刷、融合送受信を行うためのAPIを提供する。

【0080】

NCS128のプロセスは、ネットワークI/Oを必要とするアプリケーションに対して共通に利用できるサービスを提供するためのプロセスであり、ネット

ワーク側から各プロトコルによって受信したデータを各アプリケーションに振り分けたり、アプリケーションからデータをネットワーク側に送信する際の仲介を行う。具体的には、ftpd、httpd、lpd、snmpd、telnetd、smtpdなどのサーバデーモンや、同プロトコルのクライアント機能などを有している。

【 0 0 8 1 】

OC S 1 2 6 のプロセスは、オペレータ（ユーザ）と本体制御間の情報伝達手段となるオペレーションパネル（操作パネル）の制御を行う。OC S 1 2 6 は、オペレーションパネルからキー押下をキーイベントとして取得し、取得したキーに対応したキーイベント関数をSC S 1 2 2 に送信するOC S プロセスの部分と、アプリケーション130またはコントロールサービスからの要求によりオペレーションパネルに各種画面を描画出力する描画関数やその他オペレーションパネルに対する制御を行う関数などが予め登録されたOC S ライブラリの部分とから構成される。このOC S ライブラリは、アプリケーション130およびコントロールサービスの各モジュールにリンクされて実装されている。なお、OC S 1 2 6 のすべてをプロセスとして動作させるように構成しても良く、あるいはOC S 1 2 6 のすべてをOC S ライブラリとして構成しても良い。

【 0 0 8 2 】

アプリケーション130は、ページ記述言語（PDL）、PCLおよびポストスクリプト（PS）を有するプリンタ用のアプリケーションであるプリンタアプリ111と、コピー用アプリケーションであるコピーアプリ112と、ファクシミリ用アプリケーションであるファックスアプリ113と、スキャナ用アプリケーションであるスキャナアプリ114と、ネットワークファイル用アプリケーションであるネットファイルアプリ115と、工程検査用アプリケーションである工程検査アプリ116とを有している。これらの各アプリは、その起動時にVAS 140 に対して自プロセスのプロセスIDとともにアプリ登録要求メッセージを送信し、アプリ登録要求メッセージを受信したVAS 140 によって、起動したアプリに対する登録が行われるようになっている。

【 0 0 8 3 】

アプリケーション130の各プロセス、コントロールサービスの各プロセスは

、関数呼び出しとその戻り値送信およびメッセージの送受信によってプロセス間通信を行いながら、コピー、プリンタ、スキャナ、ファクシミリなどの画像形成処理にかかるユーザサービスを実現している。

【 0 0 8 4 】

このように、実施の形態 1 にかかる複合機 1 0 0 には、複数のアプリケーション 1 3 0 および複数のコントロールサービスが存在し、いずれもプロセスとして動作している。そして、これらの各プロセス内部には、一または複数のスレッドが生成されて、スレッド単位の並列実行が行われる。そして、コントロールサービスがアプリケーション 1 3 0 に対し共通サービスを提供しており、このため、これらの多数のプロセスが並列動作、およびスレッドの並列動作を行って互いにプロセス間通信を行って協調動作をしながら、コピー、プリンタ、スキャナ、ファクシミリなどの画像形成処理にかかるユーザサービスを提供するようになっている。また、複合機 1 0 0 には、サードベンダなどの第三者がコントロールサービス層の上のアプリケーション層に新規アプリ 1 1 7、1 1 8 を開発して搭載することが可能となっている。図 1 では、この新規アプリ 1 1 7、1 1 8 を搭載した例を示している。

【 0 0 8 5 】

なお、実施の形態 1 にかかる複合機 1 0 0 では、複数のアプリケーション 1 3 0 のプロセスと複数のコントロールサービスのプロセスとが動作しているが、アプリケーション 1 3 0 とコントロールサービスのプロセスをそれぞれ単一の構成とすることも可能である。また、各アプリケーション 1 3 0 は、アプリケーションごとに追加または削除することができる。

【 0 0 8 6 】

図 2 は、実施の形態 1 にかかる複合機 1 0 0 の V A S 1 4 0 の構成と、V A S 1 4 0 と各アプリ、コントロールサービス層 1 5 0 および汎用 O S 1 2 1 との関係を示すブロック図である。なお、図 2 では、アプリケーション 1 3 0 の例として、プリンタアプリ 1 1 1、コピーアプリ 1 1 2、新規アプリ 1 1 7、1 1 8 を示しているが、他のアプリでも同様の構成である。

【 0 0 8 7 】

仮想アプリケーションサービス（VAS）140のプロセスには、ディスパッチャ145と、制御スレッド144と、バージョン情報取得スレッド143と、全体バージョンチェックスレッド142と、関数単位バージョンチェックスレッド141とが動作している。

【0088】

ディスパッチャ145は、アプリケーション130やコントロールサービスからのメッセージ受信を監視し、受信したメッセージに応じて制御スレッド144、バージョン情報取得スレッド143、全体バージョンチェックスレッド142、関数単位バージョンチェックスレッド141に処理要求を行うものである。実施の形態1の複合機100では、ディスパッチャ145は、VAS140によって各アプリを仮起動させたり、通常起動したときにアプリとの間で行われるプロセス間通信を、制御スレッド144を介して各スレッド141～143間でやり取りすることができる。

【0089】

制御スレッド144は、ディスパッチャ144を介して送られてくる各アプリが使用する関数単位毎のバージョン情報や各アプリの全体バージョン情報などをバージョン情報取得スレッド143、全体バージョンチェックスレッド142、関数単位バージョンチェックスレッド141にそれぞれ渡したり、各スレッド141～143間における処理順序を制御したり、各スレッド141～143からの処理要求をディスパッチャ145に伝えたりする。

【0090】

また、関数単位バージョンチェック手段としての関数単位バージョンチェックスレッド141は、アプリが使用する関数単位毎のバージョン情報と、VAS140のインタフェースの全関数単位毎のバージョン情報とを取得して、対応する関数同士のバージョンを比較して、同じか否かをチェックするものである。

【0091】

図3は、実施の形態1におけるアプリとVASのインタフェースとの間のバージョンチェック状態を説明する図である。図3に示すように、アプリとVASをメイクする場合は、それぞれのソースプログラムをコンパイルしてオブジェクト

プログラムとする。そして、両者が共通に使用するインタフェースの関数をインクルードファイル（図中の `apiversion.h301`）とし、オブジェクトプログラムの先頭部分にインクルードして、そのオブジェクトプログラムをリンクさせることにより、実行形式のプログラムを作成することができる。このため、アプリと VAS のヘッダに組み込まれているインタフェースの関数（ここでは、API）の全体バージョンは、当初（1.0.0）と同じである。

【0092】

しかし、本発明では、バージョンチェックを全体バージョン同士を単純比較するのではなく、関数単位（API 単位）のバージョンを比較し、かつ、アプリが使用している関数についてのみ比較することにより、全体バージョンが違っても動作保証ができる場合がある点に着目している（バージョンの変更や追加がアプリの使用していない API に関するものの場合）。すなわち、これを図3で見ると、アプリ側から取得するバージョン情報を当該アプリのみが使用する関数単位毎のバージョン情報（図中の使用 API バージョン 302）としたので、VAS の全関数単位毎のバージョン情報（図中の全 API バージョン 303）がバージョンアップして、全体バージョンも（1.0.0）→（1.0.1）となり（図中の全 API バージョン 304 参照）、関数単位の API No. = 251 のバージョンが（101）→（102）に変更されても、アプリが使用していない関数のバージョンであるため、整合性には影響はなく、動作保証を得ることができる。このため、全体バージョン同士を単純比較して、バージョン差がある場合は全て動作保証が得られないとする従来のバージョンチェックと比べると、サポート範囲を広げることが可能となり、VAS の再吸収範囲を実質的に拡張することができる。

【0093】

また、図2に示す全体バージョンチェック手段としての全体バージョンチェックスレッド 142 は、各アプリ毎の全体バージョン情報と、VAS 140 のインタフェースの全体バージョン情報とを取得して、全体バージョン同士が同じか否かをチェックするものである。この手段は従来と同じであるが、本発明の特徴である関数単位毎のバージョンチェックの前段処理として、まず全体バージョン同

士を比較して、全体バージョン同士が同じであれば整合性が有りと言えるので、関数単位毎のバージョンチェックを行う必要がなくなり、処理を簡略化することができる。そして、上記した関数単位毎のバージョンチェックは、全体バージョンが違っている場合にのみ行うようにする。

【 0 0 9 4 】

さらに、図2に示すバージョン情報取得手段としてのバージョン情報取得スレッド143は、上記した関数単位バージョンチェックスレッド141で必要なアプリが使用する関数単位毎のバージョン情報とV A Sのインターフェイスの関数単位毎のバージョン情報を取得したり、上記した全体バージョンチェックスレッド142で必要なアプリの全体バージョン情報とV A Sのインターフェイスの全体バージョン情報を取得するものである。このバージョン情報取得スレッド143には、種々の形態が考えられ、新規アプリのインストール時にバージョンチェックしようとする、まだアプリが起動されていないためプロセス間通信などを使って情報を取得することができない。このため、V A S 1 4 0が対象となるアプリを仮起動させて、プロセス間通信によりアプリから必要な情報を取得するものである。ここでは、少なくとも当該アプリが使用する関数単位毎のバージョン情報を取得したが、これ以外にもプロダクトID（ベンダー、アプリ、バージョンから一義的に決定される）、ベンダー名、アプリケーション名、全体バージョン、リソース情報等を併せて取得することも可能である。また、対象となるアプリが格納されるインストール対象エリアとしては、HDD、ICカード、各種メモリカード、あるいは、ネットワークに接続されたパーソナルコンピュータ（PC）などがある。

【 0 0 9 5 】

図4は、アプリケーションを仮起動処理するか通常起動処理するかを引数によって指定するメイン関数記述例の図である。上記したように、V A S 1 4 0がアプリを起動する際に、通常起動と仮起動とを容易に使い分けることができる。例えば、図4に示すように、引数i（iオプション）を使って指定すると、仮起動がなされて、アプリ情報通知がなされる。また、引数i（iオプション）を用いない場合は、通常起動を行って、アプリ本来の動作を行うことができる。勿論、

後述するアプリ起動時、あるいは、アプリ起動後の実行時にバージョンチェックする場合は、この通常起動処理が用いられる。

【0096】

また、バージョン情報取得スレッド143は、上記以外にソースプログラムのアプリとVASをコンパイル／リンクして実行ファイルを作成する際に、少なくとも関数単位バージョンチェックスレッド141が必要とするアプリの使用関数単位毎のバージョン情報をテーブル化した使用関数テーブル212をVAS140へ通知したり、VAS140のインタフェースの全関数のバージョン情報をテーブル化した全関数テーブル211をRAM210などのメモリ上に作成し、必要に応じて利用できるようにすることが考えられる。

【0097】

次に、動作について説明する。図5は、アプリインストール時における仮起動のシーケンスを説明するフローチャートであり、図6は、図5の中のバージョンチェック処理のサブルーチンを示すフローチャートである。

【0098】

まず、図5に示すように、ユーザがオペレーションパネル上のキーやボタンを使ってアプリのインストール処理を選択すると（ステップS501）、SCS（システムコントロールサービス）122からVAS140にアプリインストール処理の開始を要求し（ステップS502）、VAS140がインストール対象エリア内のアプリを仮起動させる（ステップS503）。この仮起動は、インストールするアプリ内の各種情報をプロセス間通信を使ってVAS140に通知させ、本発明のバージョンチェック処理を行うものである（ステップS504）。

【0099】

ステップS504におけるバージョンチェック処理のサブルーチンは、図6に示すように、アプリから通知された情報にアプリの全体バージョン情報が含まれているか否かを判断し（ステップS601）、全体バージョン情報が含まれている場合は、VAS140のインタフェースの全体バージョンと比較し、同じであれば（ステップS602）、バージョンの不整合はなく、VAS140のサポート範囲内として動作保証する（ステップS603）。

【0100】

また、図6のステップS601およびステップS602において、それぞれNOが選択された場合は、ステップS604に移行し、アプリが使用する関数に対応したVAS140のインタフェースの関数毎のバージョンが全て同じか否かが判断され、同じであれば上記ステップS603にてVAS140のサポート範囲内として動作保証する（ステップS603）。しかしステップS604でバージョンの異なる関数があった場合は、VAS140のサポート範囲外となるため、動作保証できない（ステップS605）。

【0101】

このように、図5のステップS504のサブルーチンでバージョンチェック処理を行った後、他にインストール処理を行うアプリが無くなるまで上記ステップS503に戻り、上記バージョンチェック処理が繰り返される（ステップS505）。

【0102】

インストール処理するアプリが無くなった場合は、ステップS506でインストール時に取得したアプリの情報に基づいてインストール画面をオペレーションパネル上などに生成し、各種情報を表示する（ステップS506）。例えば、インストール処理された複数のアプリのバージョンチェック結果に基づいて動作保証の有無を表示すれば、ユーザは、正式なインストール要求すべきアプリをインストール画面から容易に選択することができ（ステップS507）、選択されたアプリのみを最終的にインストール処理する（ステップ508）。

【0103】

上記したように、実施の形態1では、アプリのインストール時に本発明のバージョンチェック処理を行う場合は、VASによってアプリを仮起動させ、プロセス間通信を利用してバージョン情報を含むアプリに関する情報を取得して、バージョンチェック行うことができるので、動作保証されたアプリのみを選択的にインストール処理することができる。

【0104】

また、上記以外にもアプリの通常起動時を利用して本発明のバージョンチェッ

ク処理を行うことができる。アプリを仮起動させるか通常起動させるかは、図4のメイン関数記述例に示したように、引数 *i* (*i* オプション) を用いるか否かで決まり、通常起動時にも勿論プロセス間通信を使ってアプリのバージョン情報などが取得できるため、バージョンチェック処理を行うことができる。このアプリの起動時におけるバージョンチェックは、既にインストールされているアプリを初めて起動する場合などに有効である。

【0105】

さらに、実施の形態1では、上記以外に、アプリ起動後の実行時にも本発明のバージョンチェック処理を行うことができる。動作保証のないアプリを実行するとランタイム時にエラーが発生する可能性があるため、遅くとも実行時までには本発明のバージョンチェックを行う必要がある。アプリ起動後の実行時には、プロセス間通信が使えるため、アプリの通常起動時におけるバージョンチェックと同じように行っても良いが、プロセス間通信は時間がかかるため、スムーズな実行処理が行えない。

【0106】

そこで、実施の形態1では、以下の図7および図8のフローチャートを使った別の動作例について説明する。図7は、アプリとVASをメイクする際にバージョンチェックに必要な関数テーブルを生成する手順を説明するフローチャートであり、図8は、図7で生成した関数テーブルを使ってアプリの実行時にバージョンチェックを行う手順を説明するフローチャートである。

【0107】

まず、図7のステップS701では、アプリとVASを新たに作成して画像形成装置に搭載することを要する。そして、作成するアプリとVASが共通に使用するAPIをインクルードファイルとして独立させておく（ステップS702）。図3に示した `apiversion.h301` がこれに相当する。

【0108】

そして、アプリとVASのそれぞれのソースプログラムを別々にコンパイルしてオブジェクトファイルとし（ステップS703）、そのオブジェクトファイルの先頭部分に上記したインクルードファイルをインクルードする（ステップS7

04)。さらに、インクルードファイルが組み込まれたアプリとVASのオブジェクトファイルとをリンクすることにより、実行ファイルが作成される（ステップS705）。

【0109】

その後、アプリ側では、自分の使用する関数毎のバージョンが記載された使用関数テーブル212を生成し、アプリ内部に格納しておく（ステップS706）。また、VAS側では、そのインタフェースの全関数毎のバージョンが記載された全関数テーブル211を生成して、VAS内部あるいはRAM210などのメモリに格納しておく（ステップS707）。

【0110】

これに続く図8のフローチャートでは、図7のステップS706、S707で生成された使用関数テーブル212と全関数テーブル211とを使って、アプリの実行時にバージョンチェックを行っている。まず、図8では、アプリとVASのメイク後にいずれかのバージョンの更新があったか否かが問われ（ステップS801）、更新があった場合はテーブルを新たに生成し直すことにより（ステップS802）、常に最新のバージョン情報が反映された関数テーブルを維持することができる。

【0111】

そして、VAS上でアプリを実行する場合は（ステップS803）、アプリの使用関数テーブル212の内容をVASへ通知する（ステップS804）。この通知は、情報ファイルの通知としてアプリからVASへ一方的に送られるもので、プロセス間通信と比べると格段に早く、アプリの実行時に与える影響を小さくすることができる。

【0112】

VAS140は、アプリから通知される使用関数テーブル212のバージョン情報に基づいて、対応する関数単位毎のバージョンを比較し、全てのバージョンが同じか否かを判断する（ステップS805）。関数単位毎のバージョンが全て同じであれば、バージョンの不整合はなく、VAS140のサポート範囲内であるとして動作保証する（ステップS806）。また、ステップS805でバー

ヨンの異なる関数があった場合は、V A S 1 4 0 のサポート範囲外となるため、動作保証されない（ステップ S 8 0 7）。ここでは、アプリの実行時にバージョンチェックを行っているため、図 8 のフローチャートでは詳細に説明していないが、ステップ S 8 0 6 で動作保証有りと判断された場合は、アプリの実行がそのまま継続して行われる。また、ステップ S 8 0 7 で動作保証無しと判断された場合は、アプリの実行処理を中止し、オペレーションパネル上にエラー表示などを行って、ユーザにその旨を通知するなどの対処法が考えられる。

【 0 1 1 3 】

このように、実施の形態 1 にかかる複合機では、関数単位バージョンチェックスレッドにより、アプリが使用する関数単位毎のバージョン情報と V A S のインターフェースの関数単位毎のバージョン情報を取得し、対応する関数同士のバージョンが同じか否かをチェックするので、アプリが使用していない V A S のインターフェースの関数が増減、追加、あるいは削除されて、全体のバージョンが変わっても、両者間の整合性に影響を与えないため、全体バージョン同士を単純比較する場合と比べると、V A S のバージョン差の吸収範囲が広くなって、利用可能なアプリを増やすことができる。また、アプリのインストール時から起動後の実行時までの間にバージョンチェックを行うため、ランタイム時におけるエラー発生を確実に防止することができる。

【 0 1 1 4 】

また、実施の形態 1 にかかる複合機では、アプリと V A S の全体バージョン同士を比較する全体バージョンチェックスレッドをさらに備えているため、全体バージョンが異なっている場合のみ関数単位バージョンチェックスレッドにより関数単位毎のバージョンチェックをすればよいので、常に関数単位毎のバージョンチェックを行う必要がなく、バージョンチェック処理を効率良く、かつ迅速に行うことができる。

【 0 1 1 5 】

また、実施の形態 1 にかかる複合機では、バージョン情報取得スレッドにより、アプリが使用する関数単位毎のバージョン情報と V A S のインターフェースの関数単位毎のバージョン情報を取得したり、アプリの全体バージョン情報と V A

Sの全体バージョン情報を取得することが可能なため、取得したバージョン情報同士を比較するだけで、容易にバージョンチェックを行うことができる。

【 0 1 1 6 】

また、実施の形態1にかかる複合機では、バージョン情報取得スレッドにより、予め必要なバージョン情報を持たせた実行ファイルを使い、アプリの仮起動や通常起動の際のプロセス間通信により、所望のバージョン情報を取得することができるので、バージョン情報を容易に一元管理することができる。

【 0 1 1 7 】

また、実施の形態1にかかる複合機では、バージョン情報取得スレッドは、アプリの実行ファイルの生成時にアプリが使用する関数毎のバージョン情報が記載された使用関数テーブルを生成し、V A Sの実行ファイルの生成時にV A Sのインターフェイスの関数毎のバージョン情報が記載された全関数テーブルを生成して、使用関数テーブルと全関数テーブルのいずれか一方のバージョン情報を他方へ通知することにより、比較可能なバージョン情報を取得することができるため、アプリケーションを仮起動や通常起動せずにバージョンチェックすることができる。

【 0 1 1 8 】

また、実施の形態1にかかる複合機では、関数単位バージョンチェックスレッドは、アプリのインストール時、アプリの起動時、アプリの起動後の実行時のいずれかの時点でバージョンチェックを行うようにしたため、アプリのインストール状況等に応じて関数単位毎のバージョンチェックを適切な時期に行うことが可能となり、アプリのバージョン不整合によるランタイム時のエラーの発生を未然に防ぐことができる。

【 0 1 1 9 】

また、実施の形態1にかかる複合機では、関数単位バージョンチェックスレッドがアプリのインストール時にバージョンチェックする場合、V A Sがアプリを仮起動させて、アプリが使用する関数単位毎のバージョン情報を得ることができるので、新規アプリケーションをインストールする際にV A Sとの間のバージョンの整合性をチェックすることができる。

【0120】

また、実施の形態1にかかる複合機では、関数単位バージョンチェックスレッドがアプリの起動時にバージョンチェックする場合、V A S がアプリを通常起動して、そのアプリが使用する関数単位毎のバージョン情報が得られるので、アプリとV A S との間のバージョンの整合性をチェックすることができる。

【0121】

また、実施の形態1にかかる複合機では、関数単位バージョンチェックスレッドがアプリの起動後の実行時にバージョンチェックする場合、V A S がアプリを通常起動した際に取得した当該アプリが使用する関数単位毎のバージョン情報に基づいて、その後実行される関数単位毎のバージョンをチェックするため、アプリの起動後の実行時にバージョンチェックを行うことができる。

【0122】

また、実施の形態1にかかる複合機では、関数単位バージョンチェックスレッド、全体バージョンチェックスレッド、および、バージョン情報取得スレッドの少なくとも1つがV A S に含まれるようにしたため、バージョンチェック処理を主にV A S 側で行うことができる。

【0123】

また、実施の形態1にかかる複合機では、関数単位バージョンチェックスレッド、全体バージョンチェックスレッド、および、バージョン情報取得スレッドの少なくとも1つがV A S のプロセス内部でスレッドとして生成されるため、複数のアプリを並列に起動したり、バージョン情報取得処理と関数単位バージョンチェック処理と全体バージョンチェック処理との並列処理を円滑に行うことができる。

【0124】

また、実施の形態1にかかる複合機では、関数単位バージョンチェックスレッド、全体バージョンチェックスレッド、および、バージョン情報取得スレッドの少なくとも1つがアプリケーションに含まれるようにしたため、バージョンチェック処理を主にアプリ側で行うことができる。

【0125】

また、実施の形態 1 にかかる複合機では、関数単位バージョンチェックスレッド、全体バージョンチェックスレッド、および、バージョン情報取得スレッドの少なくとも 1 つがアプリケーションのプロセス内部でスレッドとして生成されるので、これらの並列処理を円滑に行うことができる。

【 0 1 2 6 】

(実施の形態 2)

実施の形態 1 にかかる複合機 1 0 0 は、V A S 1 4 0 が全アプリケーションに対して 1 つのみ存在するものであった、この実施の形態 2 にかかる複合機では、各アプリごとに一つの V A S が起動し、各 V A S は対応するアプリとの間でバージョンチェック処理などが行われる。

【 0 1 2 7 】

図 9 は、実施の形態 2 にかかる複合機 9 0 0 の構成を示すブロック図である。図 9 に示すように、複合機 9 0 0 では、複数の仮想アプリケーションサービス (V A S) 9 4 1 ~ 9 4 8 がアプリケーション 1 3 0 の各アプリごとに動作している点が、実施の形態 1 にかかる複合機 1 0 0 と異なっている。

【 0 1 2 8 】

V A S 9 4 1 ~ 9 4 8 は、プリンタアプリ 1 1 1、コピーアプリ 1 1 2、ファックスアプリ 1 1 3、スキャナアプリ 1 1 4、ネットファイルアプリ 1 1 5、工程検査アプリ 1 1 6、新規アプリ 1 1 7 および 1 1 8 に対応して、バージョンチェック処理を行うようになっている。

【 0 1 2 9 】

図 1 0 は、実施の形態 2 にかかる複合機 9 0 0 の V A S 9 4 1 ~ 9 4 8 の構成と、V A S 9 4 1 ~ 9 4 8 と各アプリ、コントロールサービス層 1 5 0 および汎用 O S 1 2 1 との関係を示すブロック図である。なお、図 1 0 では、アプリケーション 1 3 0 として、プリンタアプリ 1 1 1、コピーアプリ 1 1 2、新規アプリ 1 1 7、1 1 8 の例を示し、さらにこれら各アプリに対応した V A S 9 4 1、9 4 2、9 4 7 および 9 4 8 を例として示しているが、他のアプリの場合も同様の構成である。

【 0 1 3 0 】

また、実施の形態 2 にかかる複合機 9 0 0 では、実施の形態 1 の複合機 1 0 0 と異なり、図 1 0 に示すように、各 V A S 9 4 1 ~ 9 4 8 と各アプリとの間には V A S 制御プロセス（デーモン） 9 0 1 が動作している。

【 0 1 3 1 】

この V A S 制御プロセス（デーモン） 9 0 1 は、各アプリと V A S が共通に使用する A P I をインクルードファイルとして持っていて、アプリと V A S のソースプログラムを別々にコンパイルしてオブジェクトファイルとし、その先頭部分にインクルードファイルをインクルードする。そして、このアプリと V A S のオブジェクトファイルをリンクさせて対応した実行ファイルを作成する。

【 0 1 3 2 】

仮想アプリケーションサービス（V A S） 9 4 1 ~ 9 4 8 のプロセスには、デイスパッチャ 1 4 5 と、関数単位バージョンチェックスレッド 1 4 1 と、全体バージョンチェックスレッド 1 4 2 と、バージョン情報取得スレッド 1 4 3 とが動作している。

【 0 1 3 3 】

デイスパッチャ 1 4 5 は、アプリケーション 1 3 0 やコントロールサービスからのメッセージ受信を監視し、受信したメッセージに応じて関数単位バージョンチェックスレッド 1 4 1、全体バージョンチェックスレッド 1 4 2、バージョン情報取得スレッド 1 4 3 に処理要求を行うものである。実施の形態 2 の複合機 9 0 0 では、デイスパッチャ 1 4 5 は、各アプリを仮起動したり通常起動したときに、アプリとの間で行われるプロセス間通信を、V A S 制御プロセス 9 0 1 を介して各 V A S 9 4 1 ~ 9 4 8 のスレッド 1 4 1 ~ 1 4 3 との間でやり取りすることができる。

【 0 1 3 4 】

関数単位バージョンチェックスレッド 1 4 1 は、アプリが使用する関数単位毎のバージョン情報と、V A S 1 4 0 のインタフェースの全関数単位毎のバージョン情報の対応する関数同士のバージョンを比較して、同じか否かをチェックするものである。

【 0 1 3 5 】

全体バージョンチェックスレッド142は、各アプリ毎の全体バージョン情報と、VAS140のインターフェースの全体バージョン情報とを取得して、全体バージョン同士が同じか否かをチェックするものである。

【0136】

バージョン情報取得スレッド143は、上記した関数単位バージョンチェックスレッド141で必要なアプリが使用する関数単位毎のバージョン情報とVASのインターフェースの関数単位毎のバージョン情報を取得したり、上記した全体バージョンチェックスレッド142で必要なアプリの全体バージョン情報とVASのインターフェースの全体バージョン情報を取得するものである。

【0137】

このように、実施の形態2にかかる複合機900によれば、実施の形態1にかかる複合機100と同様に、アプリが使用していない関数に対応したVASのインターフェースの関数に変更、追加、あるいは削除されてバージョンアップしたとしても、両者間の整合性に影響を与えないことから、全体バージョン同士を単純比較する場合と比べて、バージョン差の吸収範囲が広くなり、利用可能なアプリケーションを増やすことができる。その上、アプリケーションが実際に使用する関数単位毎のバージョン同士を個別に比較するため、ランタイム時におけるエラー発生を確実に防止することができる。

【0138】

また、実施の形態2にかかる複合機900では、VAS941～948がアプリ毎に設けられ、各アプリを個別に仮起動または通常起動させることができるため、各アプリが使用する関数単位毎のバージョン情報を取得すると共に、対応するVASのインターフェースの関数単位毎のバージョン情報を取得し、対応した関数同士のバージョンチェックを容易に行うことができる。

【0139】

【発明の効果】

以上説明したように、請求項1にかかる発明によれば、関数単位バージョンチェック手段によって、アプリケーションのインストール時から起動後の実行時までの間に、アプリケーションが使用する関数単位毎のバージョン情報を取得する

と共に、コントロールサービスをサーバとしたクライアントプロセスとして動作し、かつアプリケーションをクライアントとしたサーバプロセスとして動作する仮想アプリケーションサービスのインターフェイスの関数単位毎のバージョン情報を取得して、対応する関数のバージョンが同じか否かをチェックするようにしたので、アプリケーションが使用していない関数に対応した仮想アプリケーションサービスのインターフェイスの関数が変更、追加、あるいは削除されてバージョンアップしたとしても、両者間の整合性に影響を与えないことから、全体バージョン同士を単純比較する場合と比べて、バージョン差の吸収範囲が広くなり、利用可能なアプリケーションを増やすことができる。また、アプリケーションが実際に使用する関数単位毎のバージョン同士を個別に比較するので、確実にランタイム時におけるエラー発生を防止することができる。

【 0 1 4 0 】

また、請求項 2 にかかる発明によれば、仮想アプリケーションサービスは、アプリケーション毎に設けられ、各アプリケーションを個別に仮起動または通常起動させるようにしたので、各アプリケーションが使用する関数単位毎のバージョン情報を取得し、これに対応した仮想アプリケーションサービスのインターフェイスの関数単位毎のバージョン情報を取得して、対応する関数同士のバージョンチェックを容易に行うことができる。

【 0 1 4 1 】

また、請求項 3 にかかる発明によれば、コントロールサービスは、ユーザサービスを提供する際に、アプリケーションの少なくとも 2 つが共通的に必要とするハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うようにしたので、複数のアプリケーションを作成する際に、共通化できる部分をコントロールサービス（プラットフォーム）として共通化することにより、それ以外の部分を作成すればよく、アプリケーションの開発が容易となり、かつ、短期間で開発することができる。

【 0 1 4 2 】

また、請求項 4 にかかる発明によれば、アプリケーションの全体バージョン情報と仮想アプリケーションサービスの全体バージョン情報とを取得して、アプリ

ケーションと仮想アプリケーションサービスの全体バージョン同士を比較する全体バージョンチェック手段をさらに備えていて、全体バージョンが異なっている場合にのみ関数単位バージョンチェック手段による関数単位毎のバージョンチェックを行うようにしたので、アプリケーションが使用する関数単位毎の個別のバージョンチェックを常に行う必要がなくなり、バージョンチェック処理を効率良く、かつ迅速に行うことができる。

【0143】

また、請求項5にかかる発明によれば、バージョン情報取得手段により、アプリケーションが使用する関数単位毎のバージョン情報と仮想アプリケーションサービスのインターフェイスの関数単位毎のバージョン情報を取得したり、あるいは、アプリケーションの全体バージョン情報と仮想アプリケーションサービスの全体バージョン情報を取得することができるので、取得したバージョン情報同士を比較するだけで、容易にバージョンチェックを行うことができる。

【0144】

また、請求項6にかかる発明によれば、バージョン情報取得手段は、予め必要なバージョン情報を持たせた実行ファイルを使って、アプリケーションを仮起動、あるいは、通常起動させた際のプロセス間通信により、所望のバージョン情報を取得するようにしたので、バージョン情報を容易に一元管理することができる。

【0145】

また、請求項7にかかる発明によれば、バージョン情報取得手段は、アプリケーションの実行ファイルの生成時にアプリケーションが使用する関数毎のバージョン情報が記載された使用関数テーブルを生成し、仮想アプリケーションサービスの実行ファイルの生成時に仮想アプリケーションサービスのインターフェイスの関数毎のバージョン情報が記載された全関数テーブルを生成して、使用関数テーブルと全関数テーブルのいずれか一方のバージョン情報を他方へ通知することにより、比較可能なバージョン情報が取得できるので、アプリケーションを仮起動や通常起動させることなくバージョンチェックを行うことができる。

【0146】

また、請求項 8 にかかる発明によれば、関数単位バージョンチェック手段は、アプリケーションのインストール時、アプリケーションの起動時、および、アプリケーションの起動後の実行時のいずれかの時点でバージョンチェックを行うようにしたので、アプリケーションのインストール状況等に応じて関数単位毎のバージョンチェックを適切な時期に行うことが可能となり、アプリケーションのバージョン不整合によるランタイム時のエラー発生を未然に防ぐことができる。

【 0 1 4 7 】

また、請求項 9 にかかる発明によれば、関数単位バージョンチェック手段がアプリケーションのインストール時にバージョンチェックする場合、仮想アプリケーションサービスがアプリケーションを仮起動させて、そのアプリケーションが使用する関数単位毎のバージョン情報を得ることができるので、未インストールの新規アプリケーションをインストールする際に、仮想アプリケーションサービスとの間のバージョンの整合性をチェックすることができる。このように、インストールされていないアプリケーションであっても仮起動させることにより、アプリケーションと仮想アプリケーションサービスとの間でプロセス間通信が可能となるので、各種情報を双方向でやり取りすることが可能となる。

【 0 1 4 8 】

また、請求項 1 0 にかかる発明によれば、関数単位バージョンチェック手段がアプリケーションの起動時にバージョンチェックする場合、仮想アプリケーションサービスがアプリケーションを通常起動させて、そのアプリケーションが使用する関数単位毎のバージョン情報を得ることができるので、インストール済みのアプリケーション、あるいは、未インストールのアプリケーションをとりあえずインストールしておいて、通常起動の際にバージョンチェックを受けることができる。この場合は、アプリケーションを通常起動させるので、アプリケーションと仮想アプリケーションサービスとの間でプロセス間通信を行うことが可能であり、各種情報を双方向でやり取りすることができる。

【 0 1 4 9 】

また、請求項 1 1 にかかる発明によれば、関数単位バージョンチェック手段がアプリケーションの起動後の実行時にバージョンチェックする場合、仮想アプリ

ケーションサービスがアプリケーションを通常起動させた際に取得した当該アプリケーションの使用する関数単位毎のバージョン情報に基づいて、その後実行される関数単位毎のバージョンをチェックするようにしたので、アプリケーションの起動後の実行時においてバージョンチェックを受けることができる。この場合は、既にアプリケーションを通常起動させているので、アプリケーションと仮想アプリケーションサービスとの間でのプロセス間通信が可能であり、各種情報を双方向でやり取りすることができる。

【 0 1 5 0 】

また、請求項 1 2 にかかる発明によれば、関数単位バージョンチェック手段、全体バージョンチェック手段、および、バージョン情報取得手段の少なくとも 1 つが仮想アプリケーションサービスに含まれているので、バージョンチェック処理を主に仮想アプリケーションサービス側で行うことができる。

【 0 1 5 1 】

また、請求項 1 3 にかかる発明によれば、関数単位バージョンチェック手段、全体バージョンチェック手段、および、バージョン情報取得手段の少なくとも 1 つが仮想アプリケーションサービスのプロセス内部でスレッドとして生成されるので、複数のアプリケーションが並列に起動されたり、バージョン情報取得処理と関数単位バージョンチェック処理と全体バージョンチェック処理とを、コンテキスト切り替えなしに CPU 占有切り替えによる並列実行を行うことが可能となり、これらの並列処理を円滑に行うことができる。

【 0 1 5 2 】

また、請求項 1 4 にかかる発明によれば、関数単位バージョンチェック手段、全体バージョンチェック手段、および、バージョン情報取得手段の少なくとも 1 つがアプリケーションに含まれているので、バージョンチェック処理を主にアプリケーション側で行うことができる。

【 0 1 5 3 】

また、請求項 1 5 にかかる発明によれば、関数単位バージョンチェック手段、全体バージョンチェック手段、および、バージョン情報取得手段の少なくとも 1 つがアプリケーションのプロセス内部でスレッドとして生成されるので、バー

ョン情報取得処理と関数単位バージョンチェック処理と全体バージョンチェック処理とを、コンテキスト切り替えなしにCPU占有切り替えによる並列実行を行うことが可能となり、これらの並列処理を円滑に行うことができる。

【0154】

また、請求項16にかかる発明によれば、関数単位バージョンチェックステップによって、アプリケーションのインストール時から起動後の実行時までの間に、アプリケーションが使用する関数単位毎のバージョン情報を取得すると共に、コントロールサービスをサーバとしたクライアントプロセスとして動作し、かつアプリケーションをクライアントとしたサーバプロセスとして動作する仮想アプリケーションサービスのインターフェイスの関数単位毎のバージョン情報を取得して、対応する関数のバージョンが同じか否かをチェックするようにしたので、アプリケーションが使用していない関数に対応した仮想アプリケーションサービスのインターフェイスの関数が増加、追加、あるいは削除されてバージョンアップしたとしても、両者間の整合性に影響を与えないことから、全体バージョン同士を単純比較する場合と比べて、バージョン差の吸収範囲が広くなり、利用可能なアプリケーションを増やすことができる。また、アプリケーションが実際に使用する関数単位毎のバージョン同士を個別に比較するので、確実にランタイム時におけるエラー発生を防止することができる。

【0155】

また、請求項17にかかる発明によれば、コントロールサービスは、ユーザサービスを提供する際に、アプリケーションの少なくとも2つが共通的に必要とするハードウェア資源の獲得要求、管理、実行制御並びに画像形成処理を行うようにしたので、複数のアプリケーションを作成する際に、共通化できる部分をコントロールサービス（プラットフォーム）として共通化することにより、それ以外の部分を作成すればよく、アプリケーションの開発が容易となり、かつ、短期間で開発することができる。

【0156】

また、請求項18にかかる発明によれば、アプリケーションの全体バージョン情報と仮想アプリケーションサービスの全体バージョン情報とを取得して、アプ

リケーションと仮想アプリケーションサービスの全体バージョン同士を比較する全体バージョンチェックステップをさらに含んでいて、全体バージョンが異なっている場合にのみ関数単位バージョンチェックステップによる関数単位毎のバージョンチェックを行うようにしたので、アプリケーションが使用する関数単位毎の個別のバージョンチェックを常に行う必要がなくなり、バージョンチェック処理を効率良く、かつ迅速に行うことができる。

【 0 1 5 7 】

また、請求項 1 9 にかかる発明によれば、バージョン情報取得ステップによって、アプリケーションが使用する関数単位毎のバージョン情報と仮想アプリケーションサービスのインターフェイスの関数単位毎のバージョン情報を取得したり、あるいは、アプリケーションの全体バージョン情報と仮想アプリケーションサービスの全体バージョン情報を取得することができるので、取得したバージョン情報同士を比較するだけで、容易にバージョンチェックを行うことができる。

【 0 1 5 8 】

また、請求項 2 0 にかかる発明によれば、バージョン情報取得ステップは、予め必要なバージョン情報を持たせた実行ファイルを使って、アプリケーションを仮起動、あるいは、通常起動させた際のプロセス間通信により、所望のバージョン情報を取得するようにしたので、バージョン情報を容易に一元管理することができる。

【 0 1 5 9 】

また、請求項 2 1 にかかる発明によれば、バージョン情報取得ステップは、アプリケーションの実行ファイルの生成時にアプリケーションが使用する関数毎のバージョン情報が記載された使用関数テーブルを生成し、仮想アプリケーションサービスの実行ファイルの生成時に仮想アプリケーションサービスのインターフェイスの関数毎のバージョン情報が記載された全関数テーブルを生成して、使用関数テーブルと全関数テーブルのいずれか一方のバージョン情報を他方へ通知することにより、比較可能なバージョン情報が取得できるので、アプリケーションを仮起動や通常起動させることなくバージョンチェックを行うことができる。

【 0 1 6 0 】

また、請求項 2 2 にかかる発明によれば、関数単位バージョンチェックステップは、アプリケーションのインストール時、アプリケーションの起動時、および、アプリケーションの起動後の実行時のいずれかの時点でバージョンチェックを行うようにしたので、アプリケーションのインストール状況等に応じて関数単位毎のバージョンチェックを適切な時期に行うことが可能となり、アプリケーションのバージョン不整合によるランタイム時のエラー発生を未然に防ぐことができる。

【 0 1 6 1 】

また、請求項 2 3 にかかる発明によれば、関数単位バージョンチェックステップでアプリケーションのインストール時にバージョンチェックする場合、仮想アプリケーションサービスがアプリケーションを仮起動させて、そのアプリケーションが使用する関数単位毎のバージョン情報を得ることができるので、未インストールの新規アプリケーションをインストールする際に、仮想アプリケーションサービスとの間のバージョンの整合性をチェックすることができる。このように、インストールされていないアプリケーションであっても仮起動させることにより、アプリケーションと仮想アプリケーションサービスとの間でプロセス間通信が可能となるので、各種情報を双方向でやり取りすることが可能となる。

【 0 1 6 2 】

また、請求項 2 4 にかかる発明によれば、関数単位バージョンチェックステップでアプリケーションの起動時にバージョンチェックする場合、仮想アプリケーションサービスがアプリケーションを通常起動させて、そのアプリケーションが使用する関数単位毎のバージョン情報を得ることができるので、インストール済みのアプリケーション、あるいは、未インストールのアプリケーションをとりあえずインストールしておいて、通常起動の際にバージョンチェックを受けることができる。この場合は、アプリケーションを通常起動させるので、アプリケーションと仮想アプリケーションサービスとの間でプロセス間通信を行うことが可能であり、各種情報を双方向でやり取りすることができる。

【 0 1 6 3 】

また、請求項 2 5 にかかる発明によれば、関数単位バージョンチェックステッ

プでアプリケーションの起動後の実行時にバージョンチェックする場合、仮想アプリケーションサービスがアプリケーションを通常起動させた際に取得した当該アプリケーションの使用する関数単位毎のバージョン情報に基づいて、その後実行される関数単位毎のバージョンをチェックするようにしたので、アプリケーションの起動後の実行時においてバージョンチェックを受けることができる。この場合は、既にアプリケーションを通常起動させているので、アプリケーションと仮想アプリケーションサービスとの間でのプロセス間通信が可能であり、各種情報を双方向でやり取りすることができる。

【 0 1 6 4 】

また、請求項 2 6 にかかる発明によれば、関数単位バージョンチェックステップ、全体バージョンチェックステップ、および、バージョン情報取得ステップの少なくとも 1 つの処理が仮想アプリケーションサービスで行われるので、バージョンチェック処理を主に仮想アプリケーションサービス側で行うことができる。

【 0 1 6 5 】

また、請求項 2 7 にかかる発明によれば、関数単位バージョンチェックステップ、全体バージョンチェックステップ、および、バージョン情報取得ステップの少なくとも 1 つの処理がアプリケーションで行われるので、バージョンチェック処理を主にアプリケーション側で行うことができる。

【図面の簡単な説明】

【図 1】

この発明の実施の形態 1 である画像形成装置の構成を示すブロック図である。

【図 2】

実施の形態 1 にかかる複合機の V A S の構成と各アプリとコントロールサービス層と汎用 O S との関係を示すブロック図である。

【図 3】

実施の形態 1 におけるアプリと V A S のインタフェースとの間のバージョンチェック状態を説明する図である。

【図 4】

アプリケーションを仮起動処理するか通常起動処理するかを引数によって指定

するメイン関数記述例の図である。

【図 5】

アプリインストール時における仮起動のシーケンスを説明するフローチャートである。

【図 6】

図 5 の中のバージョンチェック処理のサブルーチンを示すフローチャートである。

【図 7】

アプリと V A S をメイクする際にバージョンチェックに必要な関数テーブルを生成する手順を説明するフローチャートである。

【図 8】

図 7 で生成した関数テーブルを使ってアプリの実行時にバージョンチェックを行う手順を説明するフローチャートである。

【図 9】

実施の形態 2 にかかる複合機の構成を示すブロック図である。

【図 1 0】

実施の形態 2 にかかる複合機の V A S の構成と、V A S と各アプリ、コントロールサービス層および汎用 O S との関係を示すブロック図である。

【符号の説明】

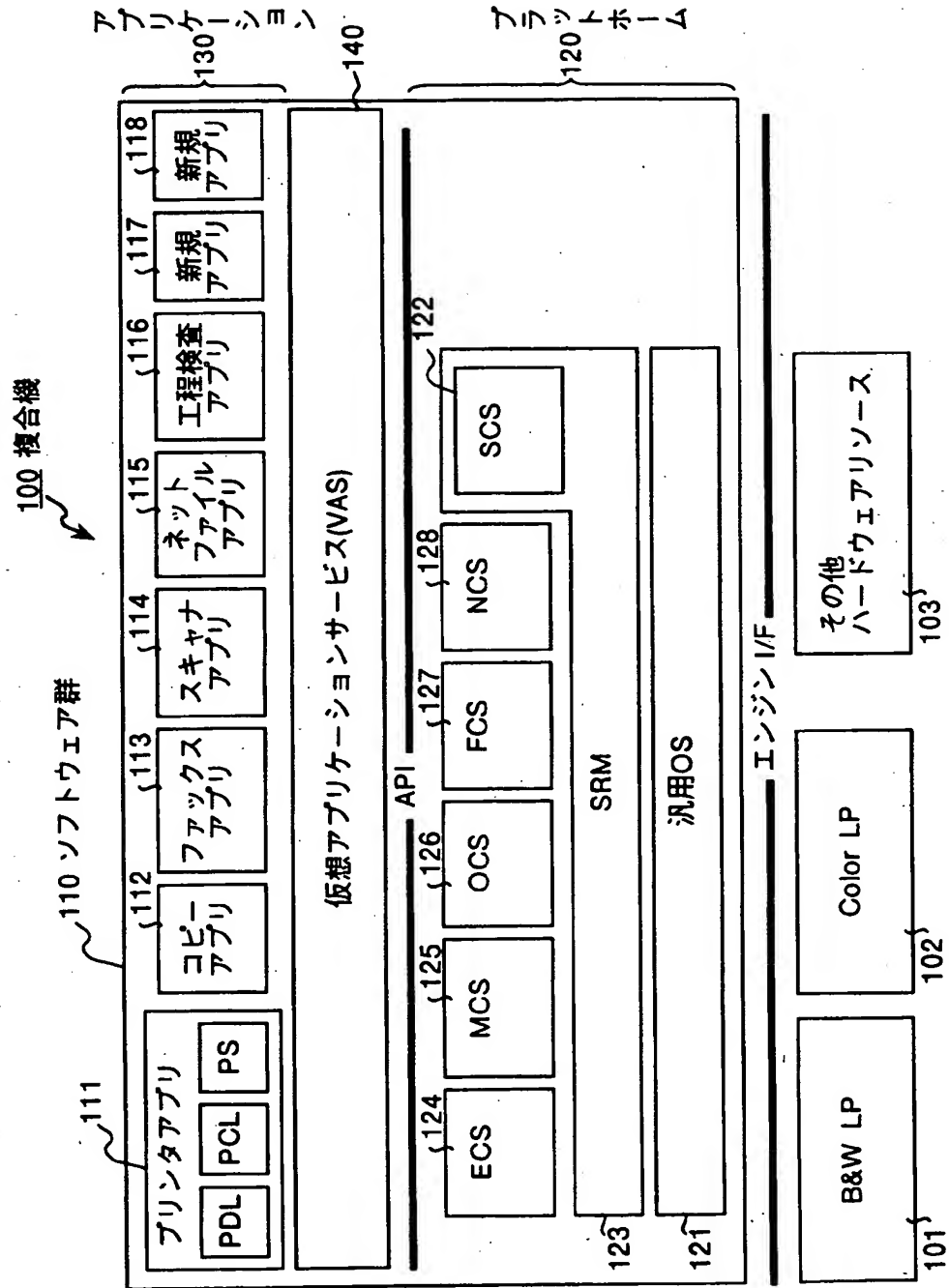
- 1 0 0 複合機
- 1 0 1 白黒レーザプリンタ
- 1 0 2 カラーレーザプリンタ
- 1 0 3 ハードウェアリソース
- 1 1 0 ソフトウェア群
- 1 1 1 プリンタアプリ
- 1 1 2 コピーアプリ
- 1 1 3 ファックスアプリ
- 1 1 4 スキャナアプリ
- 1 1 5 ネットファイルアプリ

- 1 1 6 工程検査アプリ
- 1 1 7、1 1 8 新規アプリ
- 1 2 0 プラットホーム
- 1 2 1 汎用OS
- 1 2 2 SCS
- 1 2 3 SRM
- 1 2 4 ECS
- 1 2 5 MCS
- 1 2 6 OCS
- 1 2 7 FCS
- 1 2 8 NCS
- 1 3 0 アプリケーション
- 1 4 0、9 4 1～9 4 8 仮想アプリケーションサービス (VAS)
- 1 4 1 関数単位バージョンチェックスレッド
- 1 4 2 全体バージョンチェックスレッド
- 1 4 3 バージョン情報取得スレッド
- 1 4 4 制御スレッド
- 1 4 5 ディスパッチャ
- 1 5 0 コントロールサービス層
- 2 0 1 ラッピング処理情報ファイル
- 2 1 0 RAM
- 2 1 1 全関数テーブル
- 2 1 2 使用関数テーブル
- 9 0 0 複合機
- 9 0 1 VAS制御プロセス

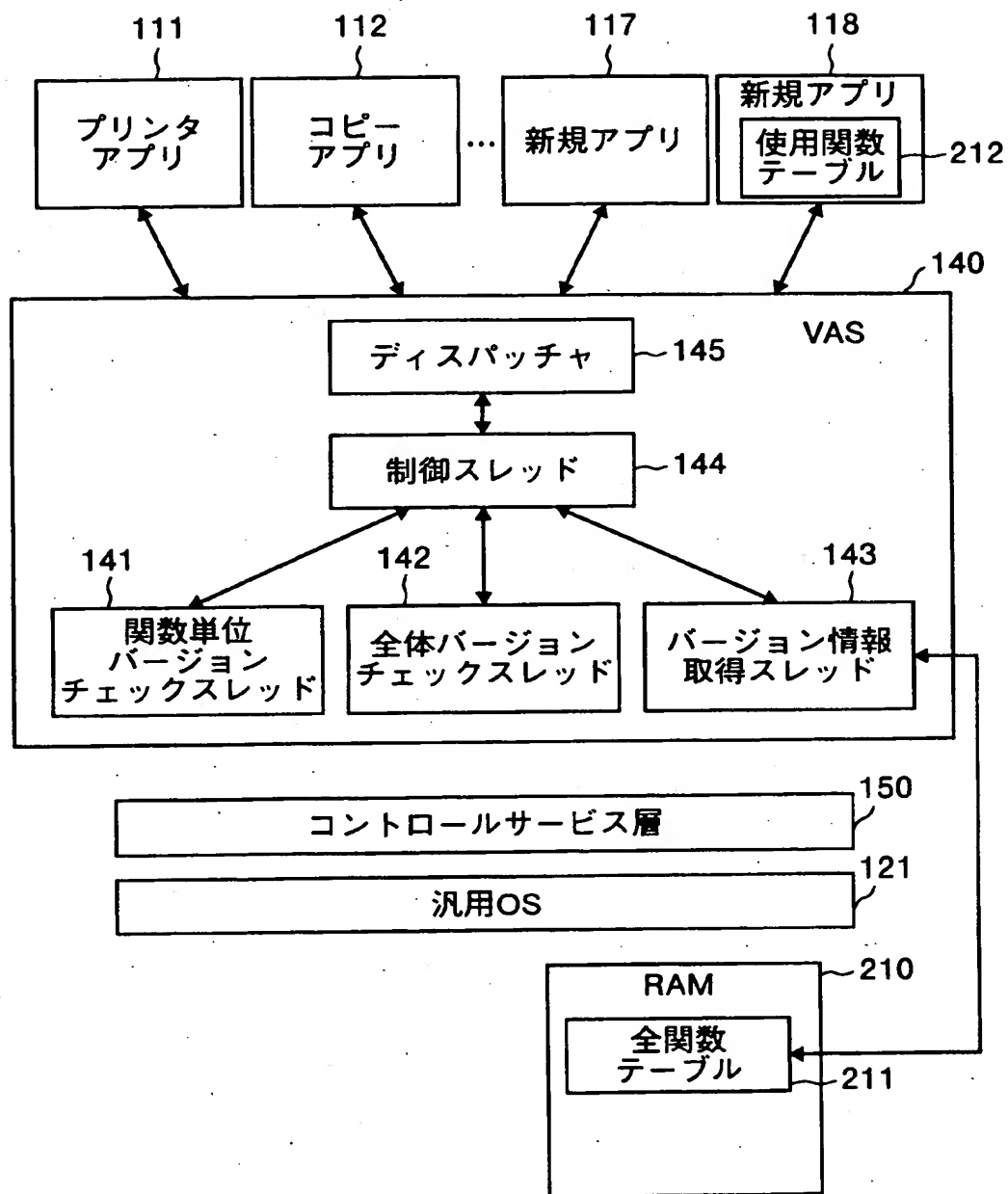
【書類名】

図面

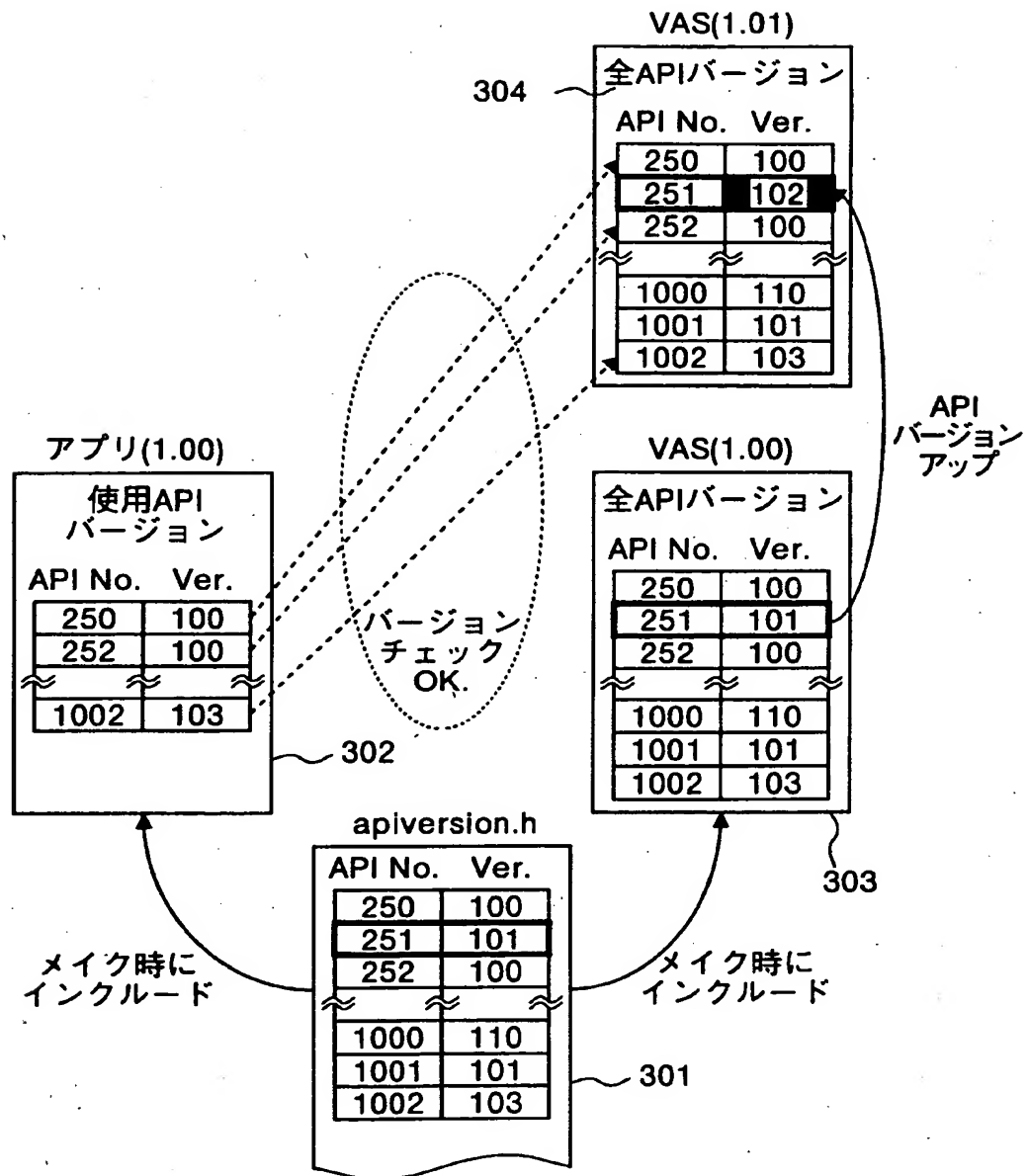
【図 1】



【図2】



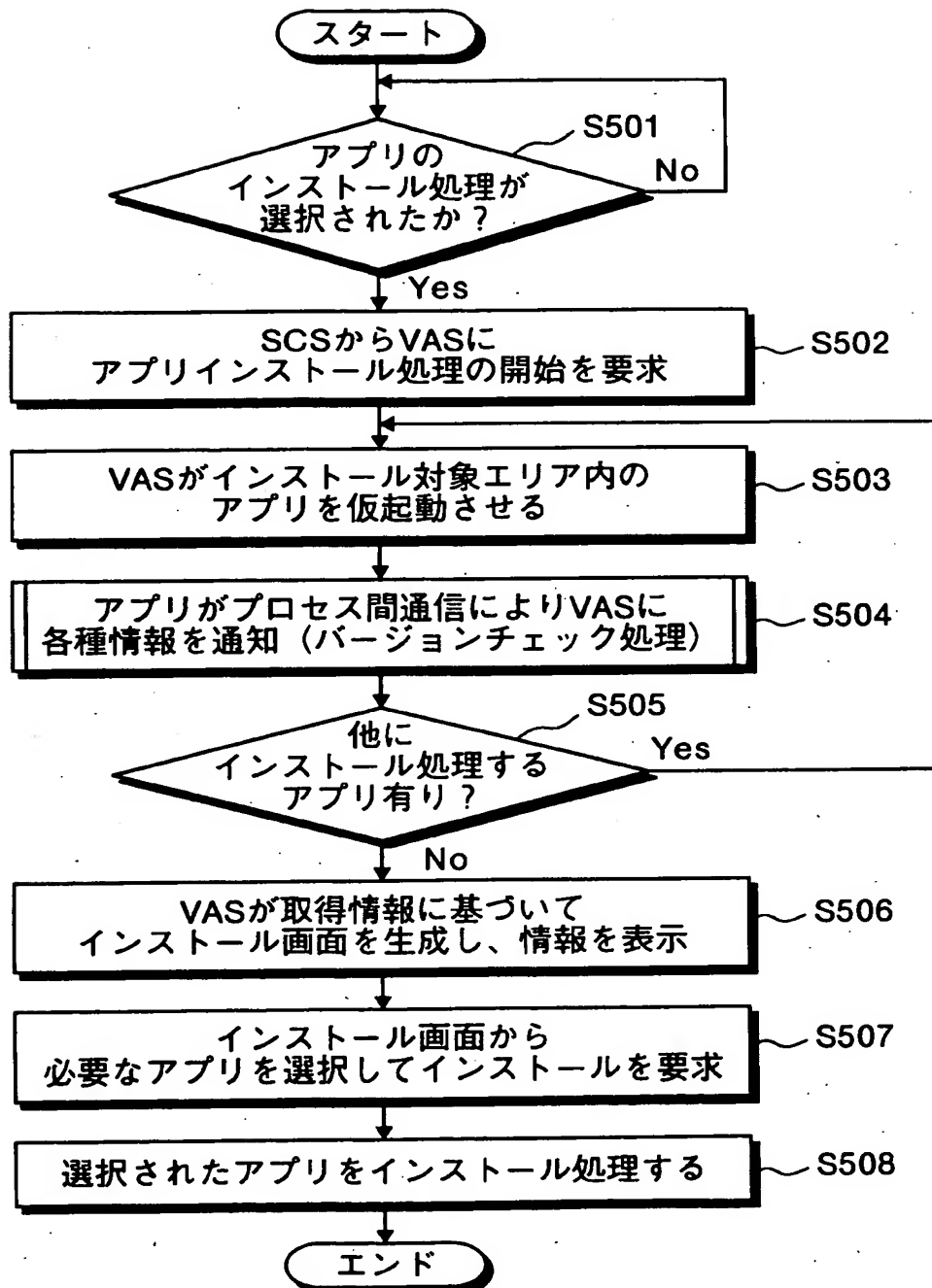
【図3】



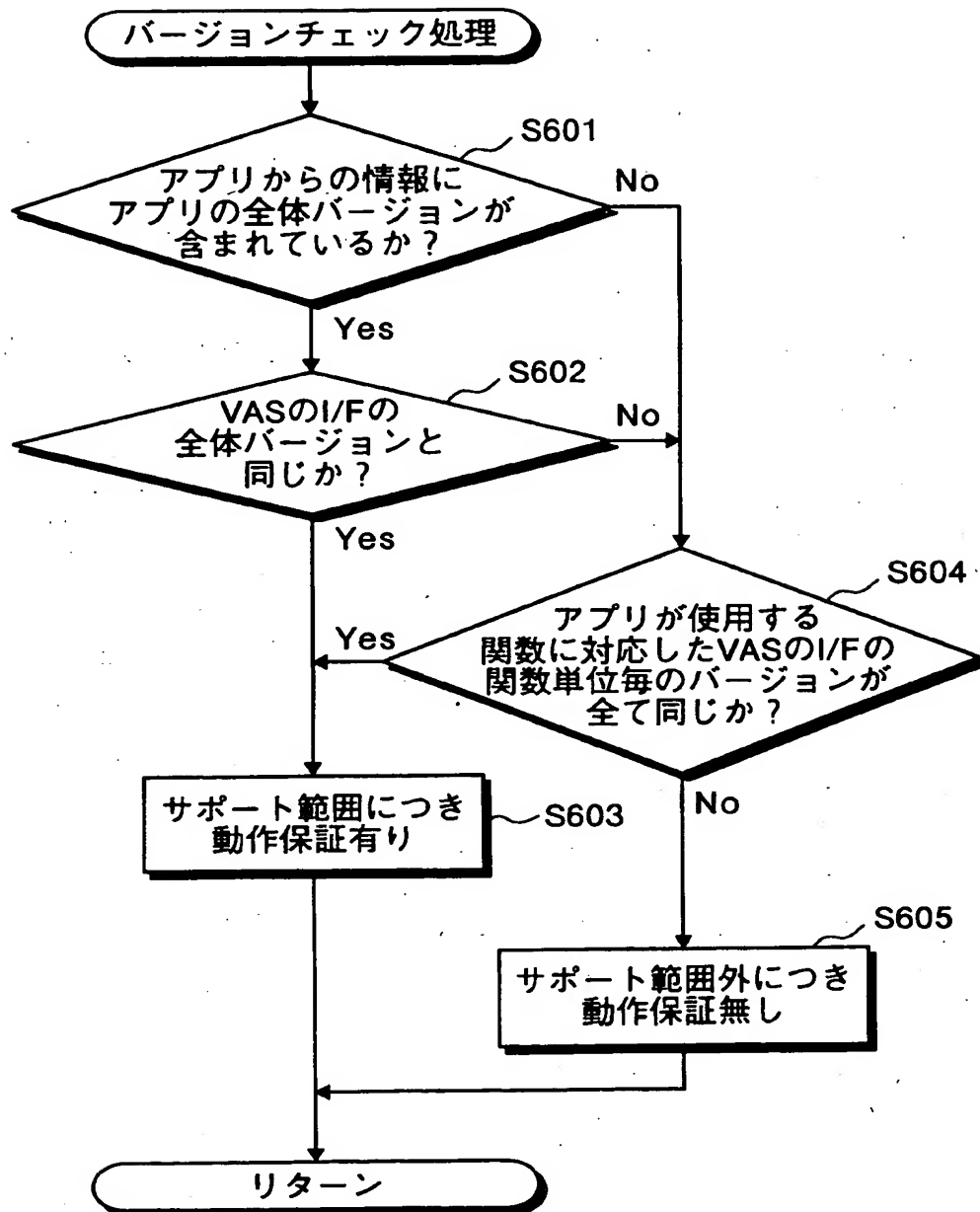
【図 4】

```
void main(int argc, char **argv)
{
    if( (argc>1)&&f(argv[1]=='i') ) {
        仮起動処理 (アプリ情報通知)
        exit(1);
    } else {
        通常起動処理 (アプリ本来の動作)
    }
}
```

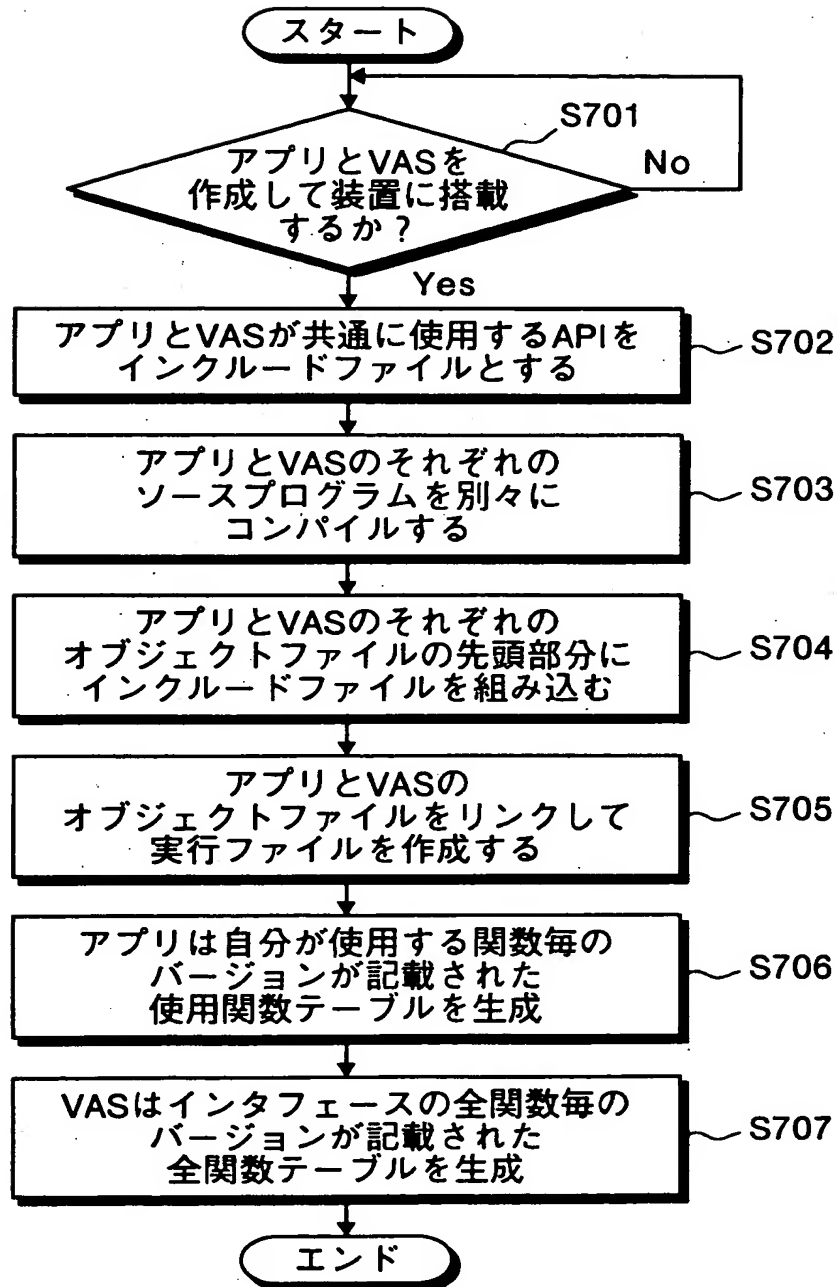
【図 5】



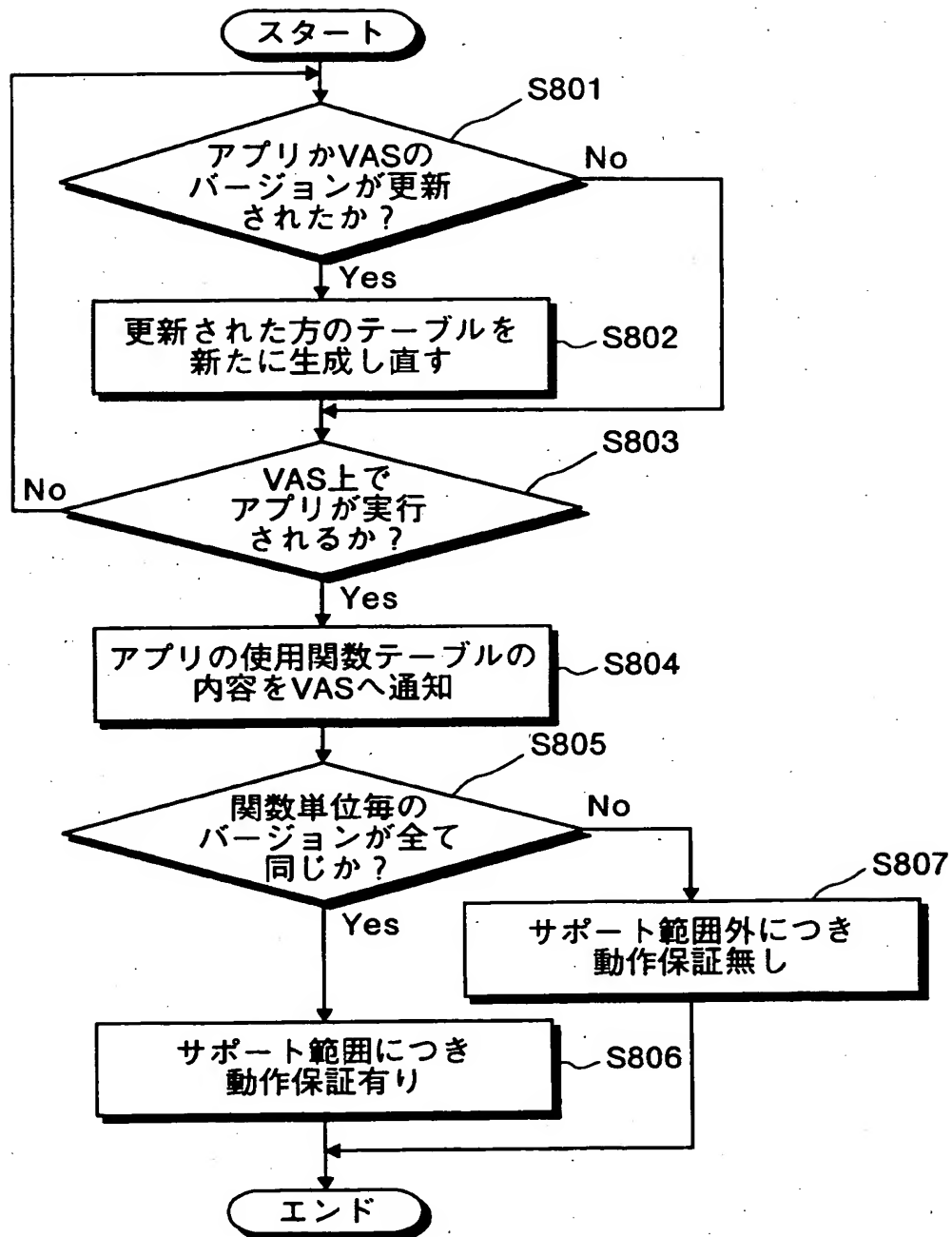
【図 6】



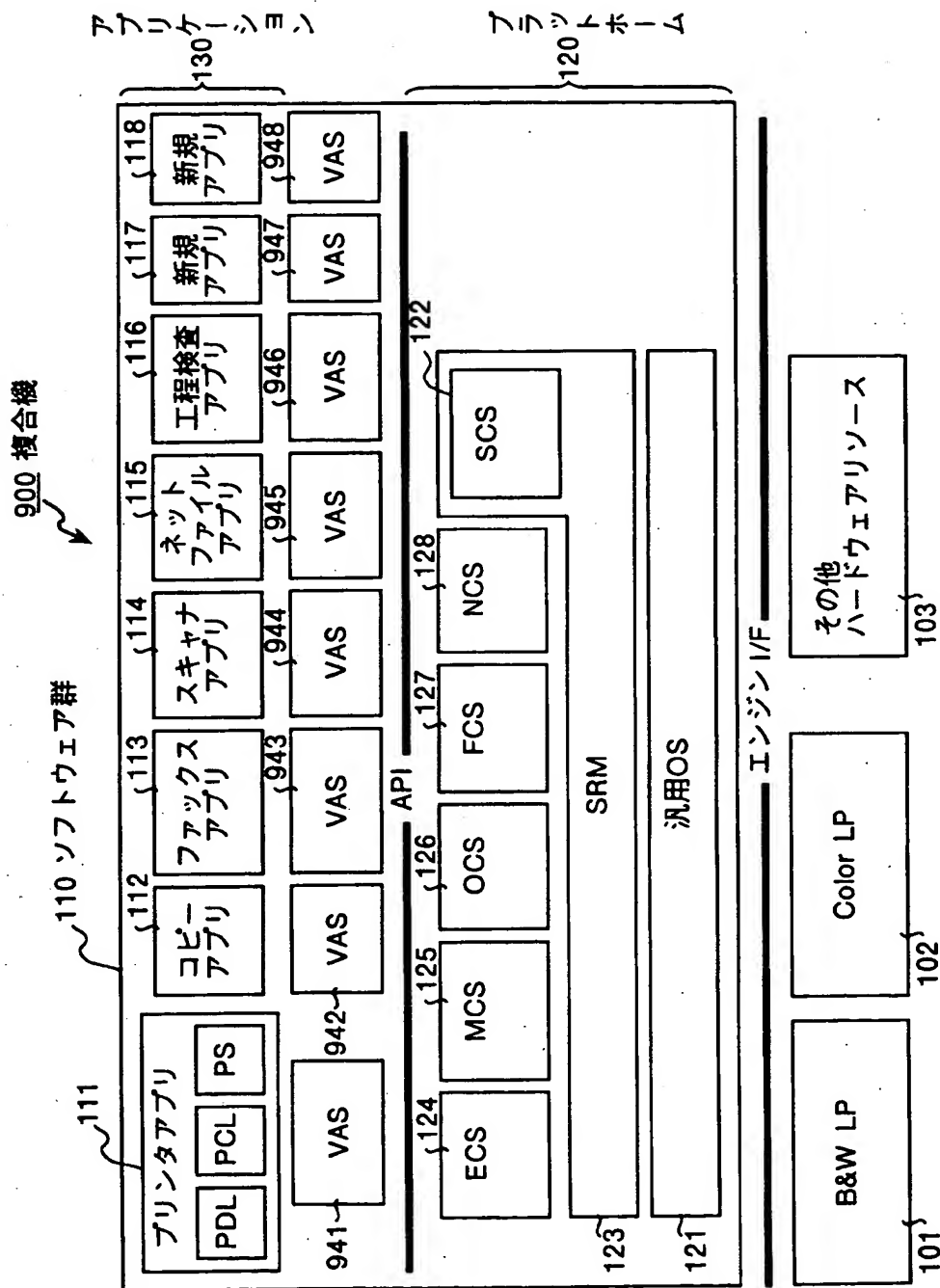
【図 7】



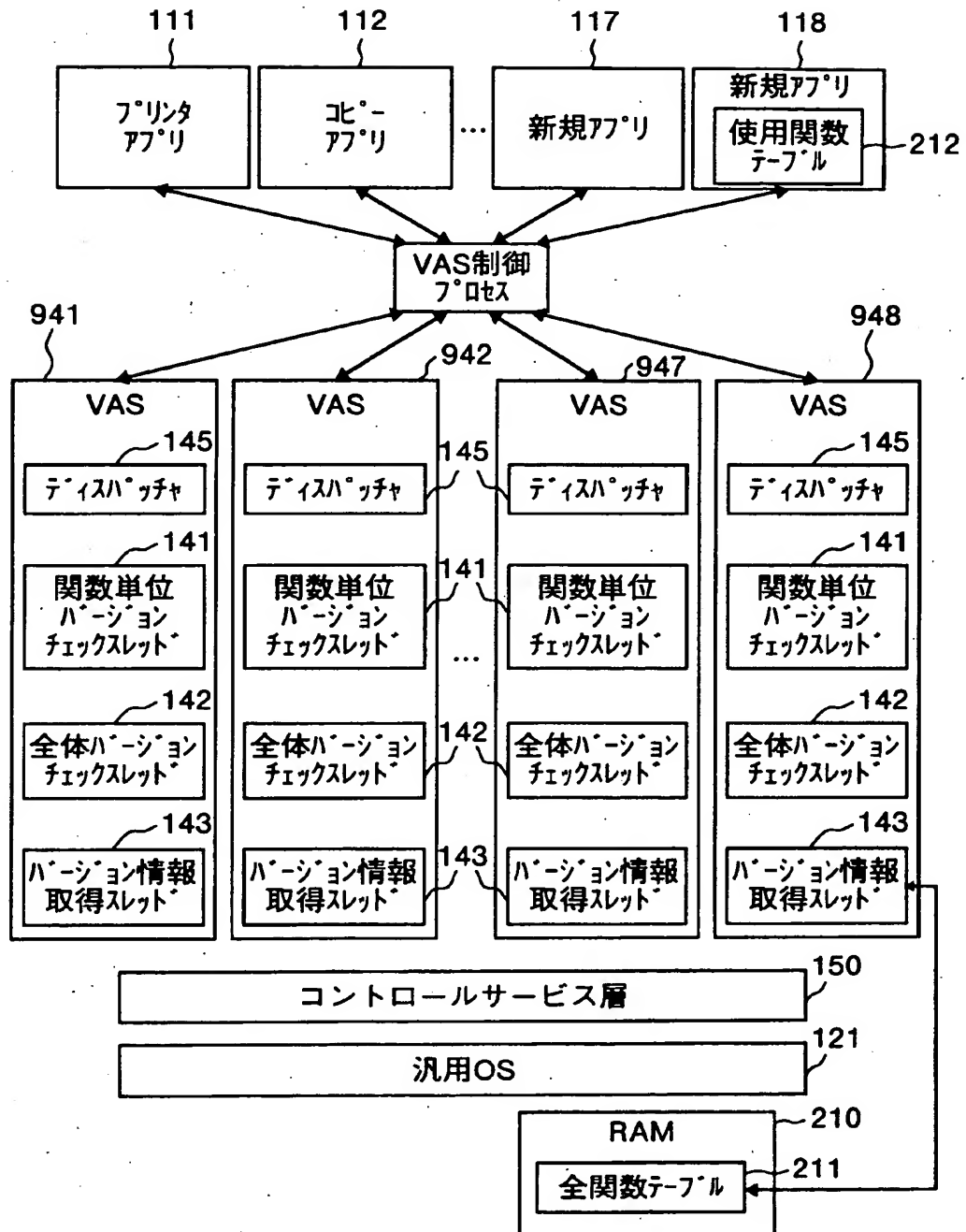
【図 8】



【図 9】



【図10】



【書類名】 要約書

【要約】

【課題】 アプリと仮想アプリケーションサービスとの間のバージョン差をできるだけ吸収して利用可能なアプリを増やし、ランタイム時のエラー発生を確実に防止するようにする。

【解決手段】 新規アプリ 1 1 8 のインストール時に、V A S 1 4 0 が新規アプリ 1 1 8 を仮起動させて、プロセス間通信を使うことにより、必要な情報を取得することができる。ここでは、新規アプリ 1 1 8 が使用する関数単位毎のバージョン情報をバージョン情報取得スレッド 1 4 3 で取得し、関数単位バージョンチェックスレッド 1 4 1 によって、V A S 1 4 0 のインタフェースの関数単位毎のバージョン情報と比較して、アプリと V A S との間のバージョン差の吸収範囲を広げることができる。

【選択図】 図 2

出 願 人 履 歴 情 報

識別番号 [000006747]

1. 変更年月日	2002年 5月17日
[変更理由]	住所変更
住 所	東京都大田区中馬込1丁目3番6号
氏 名	株式会社リコー